Green Patterns

Green Patterns

Manuel d'éco-conception des logiciels

Version 1.0



Une action Green Code Lab

MENTIONS LEGALES



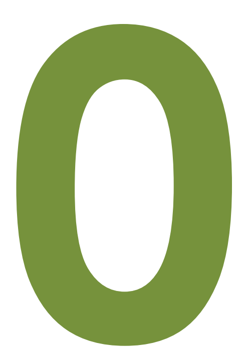
ISBN 979-10-90941-00-7

- ✓ Autres droits — d'aucune façon ne sont affectés par le contrat les droits suivants :
  - Vos droits de distribution honnête ou d'usage honnête ou autres exceptions et limitations au droit d'auteur applicables;
  - Les droits moraux de l'auteur;
  - Droits qu'autrui peut avoir soit sur l'œuvre elle-même soit sur la façon dont elle est utilisée, comme la publicité ou les droits à la préservation de la vie privée.
- ✓ Remarque — A chaque réutilisation ou distribution de cette oeuvre, vous devez faire apparaître clairement au public les conditions contractuelles de sa mise à disposition. La meilleure manière de les indiquer est un lien vers la page web http://creativecommons.org/licenses/by-nc-sa/3.0/deed.fr

Une copie de cette licence figure dans la section Annexes de ce document.

Historique de cette version

V0.1 à V1.0 : œuvre originale écrite par le groupe d'utilisateur Green Code Lab



Sommaire

1 Introduction

Collaboratif et évolutif, ce manuel se veut pratique, et à destination de toute personne du monde logiciel ou attenant.

COMMENT LIRE CE LIVRE ET Y PARTICIPER

A qui s'adresse ce livre ?

Ce livre s'adresse à toute personne qui s'intéresse à l'éco-conception des logiciels. Nous abordons ce sujet avec toujours en tête les trois piliers du développement durable : économie, social et environnement.

La première partie de ce livre aborde les problématiques générales. Une connaissance poussée du développement logiciel n'est donc pas nécessaire pour comprendre les sujets traités. La première partie, composée des chapitres 2 (Contexte) et 3 (Conception), s'adresse donc à toute personne qui a des notions de base sur le logiciel.

La deuxième partie, composée des chapitres 4 (Développement) et 5 (Après le développement), entre plus dans les détails mais sans se focaliser sur des technologies précises.

La troisième partie du livre (Chapitre 6) traite de l'éco-conception appliquée à des technologies et des domaines spécifiques. Les chapitres de cette partie nécessitent donc des connaissances des domaines traitées pour bien appréhender toutes les problématiques.

Quel est le principe du livre ?

Ce livre est un travail collaboratif et évolutif. Collaboratif car il est issu de la réflexion de plusieurs auteurs issus de différents domaines : green IT, développement logiciel, qualité logicielle, éco-conception, etc. Évolutif car la rédaction du manuscrit utilise les principes de l'agilité.

Le principe d'évolutivité permet de fournir un livre toujours d'actualité et prenant en compte les travaux actuels de l'éco-conception. En effet, ce domaine par sa jeunesse n'est pas figé et est amené à évoluer.

Par ce fonctionnement, nous souhaitons offrir aux développeurs et responsables de projet un manuel le plus complet et le plus pratique possible.

Comment contribuer à ce livre ?

Vous souhaitez faire progresser l'éco-conception des logiciels ? Ce projet en constante évolution est fait pour vous. Des encarts sont placés dans le livre par les auteurs quand des travaux ou des axes sont à développer. Si vous avez des compétences sur ces points, vous pouvez nous contacter.

La deuxième partie du livre a pour but de donner des conseils pratiques sur des technologies et des langages spécifiques (embarqué, PHP, mobile, C++...). Si vous êtes un(e) spécialiste d'un domaine qui n'est pas traité, vous pouvez aussi apporter votre expertise.

Idées d'action Green Code Lab :

Mettre en place une méthodologie d'ACV logicielle

Les encarts « Idée d'action Green Code Lab » sont des axes de travaux que nous avons identifiés et qui sont encore à traiter.

Contactez-nous sur info@greencodelab.fr

PRESENTATION DU GREEN CODE LAB

Présentation

Historique du groupe

La communauté Green IT française est active depuis plusieurs années grâce à certains acteurs clés (experts, universitaires, entreprises). En particulier greenit.fr a permis de sensibiliser les acteurs des TIC sur la problématique environnementale, de les fédérer et d'initier des actions d'amélioration. Parmi les résultats concluants on peut citer :

- ✓ « Guide pour un système d'information éco-responsable » réalisé en partenariat par GreenIT.fr, le CNRS et le WWF
- ✓ Nombreux colloques et conférences
- ✓ « Guide sectoriel TNIC » de l'ADEME
- ✓ Livre Green IT aux éditions ENI
- ✓ Livre Système d'information et développement durable, Lavoisier, 2010

Fort des constats sur les TIC et l'environnement, ainsi que sur certaines limites des actions engagées (résistance au changement des acteurs TIC, sensibilisation difficile des particuliers…), la communauté green IT a initié certaines actions pour améliorer les pratiques autour du développement logiciel :

- ✓ Mise en place d'un label sur l'éco-conception (projet en cours)

- ✓ Challenge de développement green – « Green Challenge » par Octo Technology et GreenIT.fr lors de l'USI 2010

Afin d'initier une synergie durable autour de ces pratiques, greenIT.fr soutient et participe à l'initiative Green Code Lab.

Objectifs

Green Code Lab se fixe comme objectifs de :

- ✓ Fédérer les experts du développement logiciel autour de l'éco-conception des logiciels
- ✓ Initier des bonnes pratiques en termes de développement durable
- ✓ Généraliser ces bonnes pratiques par l'alimentation d'un référentiel
- ✓ Fédérer des contenus existants en lien avec l'éco-conception pour offrir un accès simplifié à la bibliographie / échanges, bonnes idée
- ✓ Sensibiliser et former les développeurs à l'éco-conception
- ✓ Rassembler les experts issus de technologies et de domaines divers
- ✓ Favoriser des échanges entre développeurs, acteurs du développement durable, industriels et universitaires
- ✓ Travailler avec les autres communautés internationales
- ✓ Prouver la viabilité environnementale, sociale et économique des bonnes pratiques de développement et des green patterns

Esprit

Green Code Lab se veut indépendant technologiquement et politiquement. Green Code Lab a pour but de fédérer des acteurs issus de différents horizons autour d'une communauté. Cette communauté ne se veut pas uniquement virtuelle mais aussi physique et de proximité. La clé de la réussite pour cette communauté est de mêler les codes issus de plusieurs mondes : 2.0, libre, durable. L'esprit exploratoire, physique et local est donc un facteur différentiant pour Green Code Lab.

Green Code Lab s'inscrit dans un esprit collaboratif d'avant-garde inspiré par de multiples actions des communautés actuelles : TICA, java user group, cantines numériques, fablab, hackspaces… Les créateurs de Green Code Lab ont la conviction que cet esprit permettra de modifier durablement le monde du développement logiciel.

Activités

Comité et membres

Green Code Lab est une association pas comme les autres. Elle est composée d'un comité qui a pour objectif de gérer la synergie entre les groupes locaux, de donner des pistes de réflexions, de fournir l'infrastructure de fonctionnement… Ce comité est participatif et modulaire. Il n'a donc pas un rôle de direction mais plutôt d'aide. Toute personne peut faire partie du comité si elle le souhaite. Le comité se rassemble de façon opportuniste (en fonction des événements et des priorités). Il regroupe des experts issus de différents horizons : développeurs, experts développement durables…

Tous les participants actifs du groupe doivent s'inscrire en tant que membres. Cette inscription est non financière. Les membres peuvent s'inscrire au comité, participer au Green Code Lab, participer aux formations… En résumé : participer, coder, faire avancer et faire vivre Green Code Lab.

Les résultats, les données et les événements sont visibles par les non membres, l'objectif étant de partager et diffuser les résultats du groupe au plus grand nombre. Les résultats sont protégés sous licences Common Creative.

Dans l'esprit collaboratif du groupe, cette organisation est bien sûr applicable jusqu'à ce que le groupe (comité et membres) décide d'un fonctionnement encore plus durable !

Domaines traités

Green Code Lab souhaite couvrir l'ensemble de la problématique de l'éco-conception des logiciels. Tous les domaines et les technologies sont couverts :

✓ Compilé : C, C++…

✓ Interprété : .Net, Java, PHP…

✓ Base de données : SQL, noSQL…

✓ Langage intermédiaire : Ruby…

✓ Système et contraint : embarqué, noyaux GNU/Linux…

✓ Programmation parallèle

✓ Outils de développement et framework

Si nécessaire, des sous-groupes dédiés à un domaine spécifique peuvent exister. Ceci permet de se focaliser et d'avancer sur des axes sur lesquels les autres domaines n'ont pas de problème. La mise en place des bonnes pratiques nécessite une proximité par rapport aux technologies. Ses sous-groupes sont donc nécessaires. Ils sont non figés et évoluent en fonction des membres et de l'actualité. Chaque technologie et domaine possède un expert qui anime le sous-

groupe associé.

Vous vous sentez concerné ?

Vous avez un certain niveau d'expertise, vous souhaitez vous investir et promouvoir les valeurs du Green Code Lab ? Contactez-nous : info@greencodelab.fr

Evénements locaux

Green Code Lab ne se veut pas juste un groupe virtuel, les événements locaux proches des communautés existantes sont donc privilégiés. En fonction de l'importance du groupe, une section peut être créée (Sur demande des membres locaux).

Green Code Lab s'appuie sur les structures locales existantes (et communique le plus possible avec elles) pour dynamiser les actions :

- ✓ Cantines numériques
- ✓ Hackerspaces
- ✓ Groupes d'utilisateurs locaux (Java, Agile…)
- ✓ Associations et structures locales génériques

Des événements réguliers permettent aux groupes locaux de se rencontrer et d'alimenter les code labs et les bonnes pratiques. Ces événements locaux ont bien sûr pour objectif de faire participer les autres membres. La « réinvention de la roue » est une pratique commune, Green Code Lab essaie d'être innovant pour être plus efficient. Il est trop tard pour notre planète et nos enfants pour perdre du temps.

Evénements nationaux

Des événements nationaux régulier permettent de rassembler la communauté et de partager sur les retours d'événements. Les formats peuvent varier : colloque, barcamp, séminaire… Dans tous les cas, ce sont des occasions conviviales de partager dans « le monde réel » !

Le groupe sur le net

Green Code Lab a pour objectif de créer une synergie française (en relation avec les communautés internationales) autour de la mise en place de bonnes pratiques. Internet est donc un vecteur important. De plus la sensibilisation et la formation des développeurs est un des objectifs du groupe, l'accès de ces résultats est donc facilité par internet.

Le site internet permet la capitalisation et le partage des résultats par :

- ✓ Un forum
- ✓ Un code lab
- ✓ Un wiki avec des références

Green Code Lab est présent sur les réseaux sociaux : Viadeo, Linkedin, Twitter…

Le code lab

Le code lab se veut un espace d'expérimentation, et qui dit expérimentation et manipulation, dit laboratoire. Le code lab est un espace où les membres peuvent développer et partager sur les bonnes pratiques.

Le code lab est organisé en différents laboratoires. Il s'agit

d'espaces de projets où les développeurs peuvent travailler sur une application non exemplaire (Quick & dirty soft) ou sur des outils de mesure.

Les espaces sont organisés par technologies et domaines de la manière suivante :

- ✓ Bac à sable : uniquement des applications orphelines.
- ✓ Dirty soft : Environnements spécifiques pour ajouter un dirty pattern et le green pattern associé. Ceci permet de faire grossir une application (java…) et de confronter les différents Green Patterns.
- ✓ Cure d'amaigrissement : des espaces avec des applications libres ou commerciale à faire maigrir : Microsoft Windows, Firefox, GNU/Linux…
- ✓ Boite à outils : des applications ou frameworks pour aider les développeurs. Cela peut être des appli extérieures (SDX Intel Checker) ou des frameworks développés par le groupe.
- ✓ Patch for all : Parce que les pratiques découvertes dans le code lab ne sont pas uniquement destinés aux informaticiens, des scripts et programme permettant de rendre plus vert les applications existantes sont mis en place pour un accès au grand public.
- ✓ Les développeurs peuvent développer dans ce code lab seul ou en groupe via les événements organisés.

Les bonnes pratiques et formation

L'objectif principal de Green Code Lab est de mettre en place des bonnes pratiques et des green patterns. Les bonnes pratiques et green patterns sont issus des travaux dans le code lab, des événements locaux et nationaux, des échanges… Le

processus d'émergence de ces pratiques se base sur la mise en place d'axes de recherche ou d'amélioration par le comité ou par des membres. Chaque bonne pratique est validée par la communauté. Cette validation est importante car elle permet de les rendre durables et partagées par tous. Cette validation est réalisée par des développeurs ainsi que par des experts issus de domaines connexes : développement durable, économique…

Afin de capitaliser ces bonnes pratiques, des fiches pratiques, des références et des bibliographies sont rédigées sur le site par chaque membre. De plus, Green Code Lab privilégie la publication des résultats par n'importe quel moyens qui permettent la diffusion du savoir : livre blanc, livre édité, blog…

La formation et la sensibilisation des développeurs se diffusent également par le biais d'événements tels que les événements locaux réguliers.

Dans l'optique de la formation, des challenges de développement vert sont organisés pour stimuler et émuler les acteurs. Ces challenges peuvent faire intervenir des industriels, des universitaires et des développeurs.

Le sponsoring

Sponsoring des créateurs

Greenit.fr a permis la création de Green Code Lab par l'implication des experts dans la communauté green it. Ces groupes sont des groupes à but non lucratifs. Ils ne financent donc pas Green Code Lab mais participent activement à la synergie du groupe (expertise, communication…)

Sponsoring des financeurs

Le groupe nécessite un financement pour les frais de

fonctionnement, l'organisation des événements et autres frais. Ce financement est nécessaire pour que le groupe soit durable et atteigne les objectifs voulus (Changement des modes de développement, logiciel plus vert…).

Pour aller plus loin

Site : http://greencodelab.fr

Twitter : @GreenCodeLab

Contact : info@greencodelab.fr

Inscription à la newsletter :



PRESENTATION DES CONTRIBUTEURS

Présentation des auteurs

Olivier Philippot (Chef de la rédaction)

Ingénieur en électronique et informatique industrielle depuis 10 ans, Olivier Philippot a travaillé dans des laboratoires R&D et dans des grands groupes sur la conception de systèmes de gestion de l'énergie pour batteries de nouvelle génération. Ceci lui a permis, en avance de phase, de travailler sur des véhicules électriques et des engins spécifiques tels que la capsule Polar Observer de Jean-Louis Etienne. Sensibilisé tout au long de sa carrière aux contraintes utilisateurs et aux technologies d'économie d'énergie, il a mis en place le blog simplygreenit.fr pour partager son expérience sur la Green IT. Il est aujourd'hui un membre actif de la communauté Green IT française et a rejoint le blog de référence GreenIT.fr où il rédige régulièrement des articles sur les bonnes pratiques et sur l'actualité internationale Green IT. Il est l'auteur du livre « Green IT : gérez la consommation d'énergie de votre système informatique » aux éditions ENI et anime des formations et des séminaires autour de l'informatique durable.

Thierry Leboucq

Diplômé de l'EISTI Ecole Internationale des Sciences du Traitement de l'Information, Thierry a une expérience de 17 ans dans les projets SI que ce soit chez de grands donneurs d'ordre (Danone, Electrolux) qu'en tant que consultant - directeur de projet dans les sociétés de service et de conseil en SI. Habilité Bilan Carbone®, il s'est notamment spécialisé sur les outils de calcul d'impact environnemental pour l'ADEME (CoachCarbone®, Dia'terre®, base Carbone®, Base ACV®, …). En 2010, il a souhaité donner du sens à son activité

professionnelle en créant KaliTerre dans laquelle il promeut l'informatique Durable. Engagé associativement (Ouest Numérique, Accent 21, Chaire Développement Durable de l'EMN, …), il a mis son expérience des projets de développement logiciel au service de l'Éco-conception des logiciels (formation, animation de forum / tables rondes) et travaille actuellement sur la réalisation d'un outil d'évaluation environnemental d'un code informatique (projet Code Vert).

Frédéric Bordage

Diplômé de l'ESC Tours (ESCEM) et d'un DESS en informatique, Frédéric cumule 17 ans d'expérience dans le monde informatique, dont plus de 5 ans dans le Green IT. Il est reconnu comme l'un des précurseurs et meilleurs experts Green IT / TIC durables en Europe. En plus de ses missions de conseil en France et en Suisse auprès de grandes entreprises, il collabore avec différentes institutions françaises : Ademe, Afnor, Cigref, Syntec, etc. Auteur de nombreux livres sur le sujet et de plus de 30 conférences ces deux dernières années, Frédéric anime GreenIT.fr. Il a récemment écrit (avec Zen'to) le Guide sectoriel TNIC de l'Ademe. Il participe au projet PrimeEnergyIT de la Commission Européenne.

Présentation des contributeurs

Olivier Cortes :

Responsable logiciel pour la société Meta IT pour la partie retour d'expérience des couches basses.

Lionel Laské

Directeur Innovation chez C2S, la SSII interne de Bouygues et président de OLPC France. Contributeur pour les green patterns issus du challenge USI et le retour d'expérience OLPC

.

Green Software Engineering

Projet collaboratif allemand travaillant sur l'éco-conception. Contribution sur les recommandations Web.

Remerciements

Ils ont relu attentivement le manuscrit et ont apporté leurs remarques, expériences et améliorations :

Aurélien Probst

Bruno Seguin

2 Contexte

L’utilisation croissante des technologies de l’information associée à une certaine obésité du logiciel amène à se poser des questions sur l’impact indirect du code sur l’environnement. Voici une mise en situation sur le logiciel et le développement durable.

IMPACT DU LOGICIEL SUR L'ENVIRONNEMENT

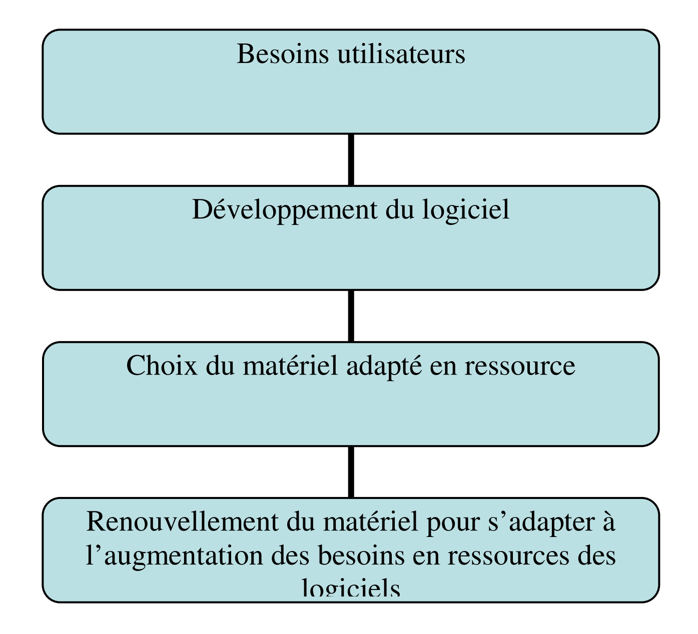
Introduction

Les logiciels sont souvent présentés comme des ressources immatérielles. Pourtant, pour pouvoir les faire fonctionner, nous utilisons de nombreux équipements électroniques, dont la fabrication, l'utilisation et la fin de vie ont un impact direct négatif certain sur l'environnement.

En effet, comment lire un livre électronique sans écran ? Comment envoyer un courriel sans ordinateur ? Comment répondre à un SMS sans téléphone ? Comment matérialiser un billet d'avion ou de train sans imprimante ? Bref, même s'ils ne constituent qu'une suite de 0 et 1 stockés sur des disques durs ou des clés USB, les logiciels ont une empreinte écologique indirecte conséquente.

Les logiciels sont mêmes la principale cause de l'obsolescence accélérée des équipements informatiques. Et donc, indirectement, la principale cause de l'augmentation continue de l'empreinte écologique de l'informatique.

En effet, il ne faut pas oublier que la chaîne logique d'un projet informatique est la suivante :



Les besoins des logiciels en ressources matérielles – quantité de mémoire vive, puissance du processeur, espace disque – conditionnent directement la fréquence de renouvellement du matériel. Or, les ressources nécessaires à l'exécution d'un logiciel doublent à chaque nouvelle version[1]. Et les éditeurs publient une nouvelle version de leur logiciel tous les 2 à 3 ans maximum[2].

Pour réduire efficacement l'empreinte écologique directe des équipements, il faut donc commencer par réduire les besoins en ressources des logiciels.

[1] GreenIT.fr, Frédéric Bordage et Frédéric Lohier, 2010
[2] GreenIT.fr, Frédéric Bordage, 2011

Les trois piliers du développement durable

Le développement durable est : « un développement qui répond aux besoins des générations présentes sans compromettre la capacité des générations futures à répondre aux leurs.»[3]

L'objectif du développement durable est de définir des schémas viables à long terme qui concilient les trois aspects économique, social, et écologique des activités humaines.

Il s'agit à la fois de :

- ✓ trouver des solutions à court terme aux crises écologiques, sociales et économiques qui se manifestent désormais partout dans le monde (changement climatique, épuisement des ressources naturelles non renouvelables, pic pétrolier, sécurité alimentaire, écroulement de la biodiversité, catastrophes naturelles et industrielles, écarts nord / sud, etc.),

- ✓ tout en proposant un modèle de développement « durable » pour l'avenir.

Afin de subvenir aux besoins actuels sans pour autant se reposer sur l'utilisation de ressources non renouvelables, un scénario en trois points a été proposé par l'ONU :

- ✓ efficience (techniques plus performantes),

- ✓ sobriété (techniques utilisées avec parcimonie),

- ✓ utilisation de ressources renouvelables.

Appliqué aux technologies de l'information et de la

[3] Commission Européenne, Rapport Bruntland, 1987

communication (TIC), le développement durable consiste à :

- ✓ réduire
  - o l'empreinte (écologique, économique, et sociale) des TIC,
  - o utiliser les TIC pour réduire l'empreinte de l'humanité,
- ✓ inventer un nouveau modèle de société (organisation économique, spatiale et sociale)
  - o en s'appuyant sur les trois piliers du développement durable
  - o en tenant compte des limites de notre écosystème
  - o en utilisant intelligemment les TIC existantes

Pour ne pas compromettre la capacité des générations futures à répondre à leurs besoins.

Dans ce contexte, les TIC durables sont généralement organisées selon trois périmètres :

- ✓ Green IT 1.0 ou Green for IT : réduire l'empreinte des TIC,
- ✓ Green IT 1.5 ou IT for Green : réduire l'empreinte de l'organisation grâce aux TIC
- ✓ Green IT 2.0 ou IT for Green : inventer de nouveaux produits et services, plus durables, grâce aux TIC.

Ces trois périmètres sont complémentaires et les organisations (entreprises / collectivités) peuvent mener ces projets en parallèles.

La figure ci-dessous présente une vue synthétique des différents périmètres des TIC durables.

Dans cet ouvrage, nous nous intéresserons principalement au premier périmètre : comment réduire l'empreinte des TIC grâce à des logiciels plus économes en ressources informatiques (cycles CPU, quantité de mémoire vive, etc.).

Pour aller plus loin

Greenit.fr : blog de référence sur la green IT

Guide pour un système d'information éco-responsable, WWF

Livre Green IT – Olivier Philippot - Aux éditions ENI

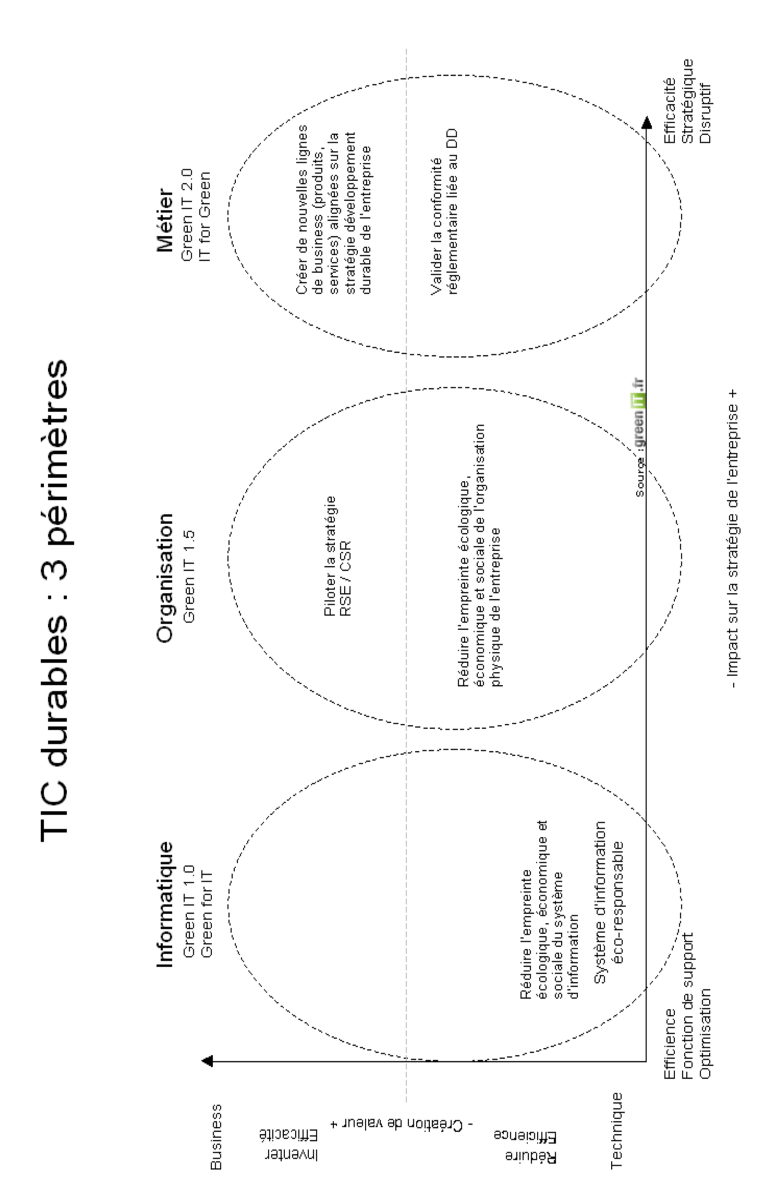


Figure 1 - Les trois périmètres des TIC durables. Source : GreenIT.fr, 2010, Frédéric Bordage

Quel impact du logiciel sur l'environnement ?

Comme nous l'avons vu dans l'introduction de ce chapitre, les besoins des logiciels - mémoire, puissance processeur, espace disque - et leur fréquence de mise à jour conditionnent la durée d'utilisation des équipements électroniques : ordinateur, serveur, tablette, smartphone, etc.

L'empreinte environnementale des technologies de l'information et de la communication (TIC) est liée principalement à la fabrication et à la fin de vie du matériel. Pour réduire cette empreinte, il faut donc utiliser le matériel le plus longtemps possible. Or, nous faisons exactement l'inverse : la durée d'utilisation d'un ordinateur a été divisée par 4 en 25 ans pour atteindre moins de 3 ans en 2005[4]. D'où une question essentielle : pourquoi renouvelons-nous prématurément le matériel informatique ?

La couche logicielle est le principal responsable de l'obsolescence accélérée des ordinateurs, des tablettes numériques et des smartphones. Essentiellement à cause des besoins en ressources (mémoire vive, cycles CPU, espace disque, etc.) des nouvelles versions de logiciels qui augmentent constamment. Aucun responsable informatique ne peut prendre le risque de fonctionner sans support et maintenance technique. Il met donc à jour les logiciels dès que le support technique s'achève, au bout de 3 à 5 ans en moyenne.

Or, chaque nouvelle version nécessite, en moyenne, 2 fois plus de puissance[5] que la version précédente. Si on prend l'exemple de Microsoft, chaque nouvelle version du couple Microsoft Windows-Office nécessite 2 fois plus de ressources que la

4 GreenIT.fr, août 2010, Compilation de 3 études scientifiques (E. Williams, EPA, et Seikatsu Jouhou Center: Tokyo, 2002)

5 GreenIT.fr, 2011, Fred Bordage

précédente. En d'autres termes, la puissance nécessaire pour écrire un texte double tous les deux ans[6]. Si bien qu'il faut 70 fois plus de mémoire vive sous Microsoft Windows 7 - Office 2010 pour écrire le même texte que sous Microsoft Windows 98 - Office 97 ! On imagine mal devoir utiliser une voiture 70 fois plus puissante qu'il y a 12 ans pour parcourir le même nombre de kilomètres, à la même vitesse. C'est pourtant ce qui se passe dans l'industrie du logiciel.

L'impact environnemental lié à cette frénésie est simple : 3 fois plus d'ordinateurs fabriqués ces 25 dernières année que nécessaire. Or, sur les 24 kg de déchets d'équipements électriques et électroniques (DEEE) émis chaque année par un français, nous n'en collectons que 5 kg. La quantité de DEEE augmente plus vite que notre capacité à les retraiter. Entre 2006 et 2009, 370 506 tonnes d'équipements électriques et électroniques (EEE) professionnels de catégorie 3 (matériel informatique) ont été mis sur le marché, mais seulement 51 579 tonnes de DEEE professionnels de catégorie 3 ont été collectées et retraitées . Cette situation est problématique car les DEEE sont des sources importantes de pollutions et d'épuisement des ressources non renouvelables. Le manque d'infrastructures de traitement des DEEE entraîne la disparition d'importantes quantités de cuivre, d'or, d'argent, de palladium , d'indium et autres ressources rares. La récupération de ces matériaux limiterait pourtant la pression sur les écosystèmes.

Mais plus encore que l'épuisement des ressources non renouvelables, les TIC concentrent de nombreuses substances chimiques nocives pour l'environnement et la santé. La directive européenne RoHS restreint – sans les interdire – la quantité pouvant être utilisée dans les équipements électroniques pour les six substances suivantes : le mercure, le plomb, le cadmium, le chrome hexavalent, les polybromobiphényles (PBB), et polybromodiphényléthers (PBDE). Mais cette réglementation est loin d'être suffisante. D'autres substances préoccupantes continuent à être utilisées

[6] GreenIT.fr, 2010, Fred Bordage et Frédéric Lohier

par les industriels malgré le risque qu'elles représentent : les retardateurs de flammes halogénés, les additifs du PVC, les phtalates, le bisphénol A, l'arsenic, etc. Selon l'US Environmental Protection Agency, 70% des métaux lourds (extrêmement toxiques pour les êtres vivants) présents dans les décharges nord-américaines proviennent du matériel électronique qui s'y accumule. En s'infiltrant dans le sol, puis dans les nappes phréatiques, ces substances toxiques remontent la chaîne alimentaire… jusque dans notre assiette…

Internet, cloud computing, logiciel et environnement

On présente souvent le cloud computing (informatique dans le nuage) comme une alternative éco-responsable à l'architecture technique actuelle qui repose encore souvent sur le modèle client-serveur. Hérité de l'ère de la micro-informatique personnelle et de la bureautique, le modèle client-serveur nécessite un poste de travail assez puissant pour effectuer de nombreuses opérations complexes en local.

A l'inverse, le modèle du cloud computing part du principe que la majorité des traitements lourds sont effectués sur les serveurs et que le poste de travail n'est plus qu'un terminal connecté. Dans cette approche, la plus grosse part de l'empreinte environnementale se concentre dans les data centers qui hébergent les services en ligne et dans l'infrastructure réseau permettant de se connecter quasiment n'importe où via une liaison xDSL ou 3G.

L'objectif du cloud computing est de mutualiser les moyens informatiques en interne (cloud privé) ou en externe (cloud public) pour réduire le coût de fonctionnement et l'empreinte écologique. La plupart des opérateurs de nuages s'appuient sur des data centers récents qui propose un faible PUE (Power Usage Effectiveness). De ce point de vue, les nuisances environnementales liées à la phase d'utilisation paraissent, en théorie, plus faible que le modèle informatique traditionnel. La

mutualisation du matériel devrait également réduire l'empreinte écologique d'un service si l'on considère l'intégralité de son cycle de vie.

Pourtant, on note déjà plusieurs effets rebond qui, dans bien des cas, annihilent les gains théoriques du cloud computing. Le premier effet rebond est lié au niveau de service (SLA) demandé par les entreprises qui migrent leur infrastructure interne dans le nuage. Elles demandent généralement un SLA supérieur à celui qu'elles offraient en interne. Or, pour fournir un SLA plus élevé, il faut généralement redonder l'infrastructure du data center pour atteindre un niveau tier3+ ou tier4. La redondance du matériel se traduit par une plus forte empreinte écologique.

Par ailleurs, le coût de fonctionnement d'un data center est désormais principalement lié à la consommation électrique et aux tâches d'administration qu'à l'investissement dans les serveurs. Selon une étude d'IDC, il faut en moyenne 1 900 $ par an pour alimenter et refroidir un serveur qui a coûté 2 500 $. Cette situation économique pousse les opérateurs de data center à renouveler leurs équipements bien avant leur fin de vie électronique, pour faire des économies. Dès lors, la capacité du cloud computing à réduire l'empreinte écologique d'un logiciel ne tient plus.

ÉVOLUTION DU LOGICIEL

Loi de Moore

Pour comprendre l'évolution du logiciel en terme d'impact environnemental et social, il faut se poser plus globalement la question : Où va la technologie ? Bien futé celui qui le prédira. Cependant, depuis 1965, une loi prédit correctement l'évolution du logiciel : la loi de Moore. Elle se vérifie année après année. La Loi de Moore est simple : « Quelque chose » (le nombre de processeurs, la performance, la vitesse…) double tous les 18 mois. Cette loi a été critiquée, reprise, interprétée et modifiée, mais toujours vérifiée.

Depuis plusieurs années, les limites de cette loi semblent être atteintes. Mais chaque barrière technologique a été dépassée. Il y a dix ans on annonçait la fin de la croissance linéaire de la puissance des microprocesseurs, mais les technologies multi-cœurs ont permis de continuer cette folle croissance.

Nous n'allons pas discuter ici de la véracité et du futur de la loi de Moore, ce sujet est largement discuté dans la littérature. La question importante, est : pourquoi cette loi stimule-t-elle de façon si importante le monde des technologies depuis les 30 dernières années ?

Sûrement parce qu'avant d'être une loi technologique, c'est une loi économique. C'est ce que Gordon Moore appelle lui-même une prophétie auto réalisatrice. La Loi de Moore est un leitmotiv pour l'industrie électronique… qui met les moyens marketing et techniques pour atteindre les objectifs de la loi.

Quel est alors l'impact de cette loi sur l'environnement ? Y-a-t-il un corollaire "vert" ou "durable" à la loi de Moore ? Une réponse a été donnée par Jonathan Koomey du Stanford and Lawrence Berkeley National Laboratory. Selon Koomey, le nombre de

calculs par kWh doublerait tous les 18 mois. La loi de Koomey est plutôt satisfaisante car l'amélioration de l'efficience des processeurs est bien une réalité. Peut-on en déduire que tout va dans le bon sens ? Non, répondent Frédéric Lohier et Frédéric Bordage, contributeurs de GreenIT.fr : les besoins des logiciels en ressources matérielles (processeur, mémoire, etc.) doublent d'une version à l'autre[7]. Besoins qui mènent à une obsolescence accélérée du matériel. Ce phénomène se retrouve dans tous les logiciels : OS, logiciels open source, logiciels commerciaux… Les besoins en mémoire, en espace disque et en CPU explosent, mais, en même temps, les logiciels sont de plus en plus lents. Logiciel et matériel sont donc liés par un phénomène que nous appellerons le bloatware ou obésiciel.

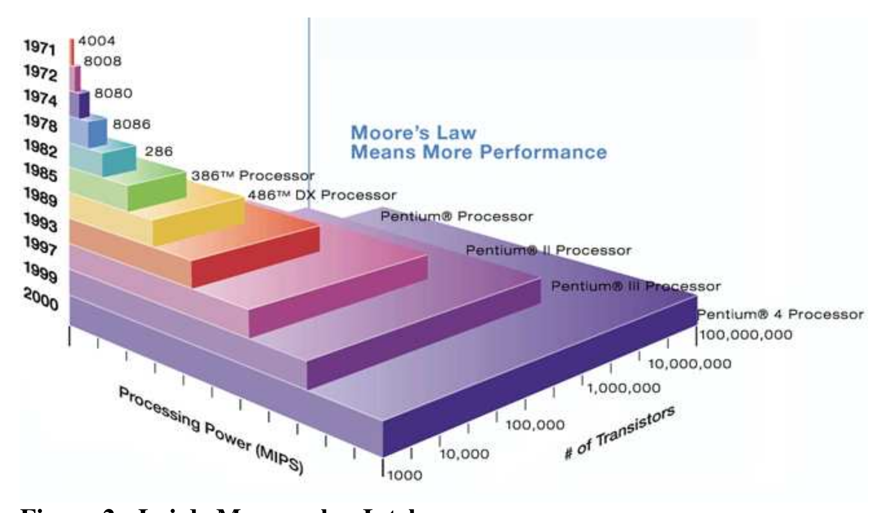


Figure 2 - Loi de Moore selon Intel

[7] Logiciel : la clé de l'obsolescence programmée du matériel informatique, 2010 - http://www.greenit.fr/article/logiciels/logiciel-la-cle-de-l-obsolescence-programmee-du-materiel-informatique-2748

Les lois du logiciel

Comment expliquer ce phénomène d'obésiciel ? Le logiciel subit-il la loi de Moore ? Peu probable. Même si on peut corréler certaines données à une croissance similaire à la loi de Moore. De plus le logiciel dépend de nombreux paramètres : langage de programmation, qualité de la programmation, processus et structure de l'équipe, projet open source… Alors existe-t-il une loi qui caractérise l'évolution du logiciel comme la loi de Moore ? Cela permettrait de dégager des axes d'amélioration. Car même si des bonnes pratiques de développement sont à mettre en place, une remise en question plus profonde est, peut-être, nécessaire. Peut-être qu'avec la meilleure des volontés les développeurs n'y pourront rien ?

Pour comprendre la complexité du bloatware, on peut se poser la question suivante sur le cas particulier de la parallélisation : Même si le matériel nous offre des capacités de paralléliser les calculs avec des processeurs multi-cœurs ultra efficaces, pourrons-nous utiliser toute la capacité de ces technologies ?

Appuyons-nous sur une loi qui essaye de décrire le comportement de l'informatique. Une loi moins connue que celle de Moore, mais toute aussi discutée est la loi de Amdahl (1967) : elle essaie de caractériser la capacité de l'informatique à traiter le parallélisme.

La loi d'Amdahl énonce que lorsque l'on parallélise un calcul, une partie ne peut être réalisée qu'en série. Pour vulgariser : ce n'est pas en mettant 9 femmes enceintes que l'on va accoucher un bébé en un mois ! Dave Probert, architecte du noyau Microsoft Windows, cite la loi d'Amdahl[8] comme une nouvelle force à dépasser dans l'innovation logicielle. Barrière technique renforcée par la multiplicité et l'hétérogénéité des processeurs multi-cœur. Encore une excuse de développeur pour ne pas se

[8] http://www.slideserve.com/presentation/23027/Evolution-of-the-Windows-Kernel-Architecture

fouler ? Peut-être pas… surtout si l'on tient compte d'une autre loi (promis, c'est la dernière) : celle de Lehman.

Que dit cette loi ? Lehman estime que l'évolution d'un logiciel est inévitable, pour différentes raisons. Il a énoncé 8 lois entre 1964 et 1996, voici 3 lois qui nous concernent :

1 – Modifications perpétuelles : Un logiciel doit constamment être adapté, sinon il devient de moins en moins utile.

2 – Complexité croissante : Au fur et à mesure que le logiciel évolue, sa structure a tendance à se complexifier.

6 – Continuelle augmentation : le contenu du logiciel doit continuellement augmenter pour satisfaire les besoins des utilisateurs.

Une étude sur le projet Firefox confirme en partie ces lois[9].

Est-ce encore une fois une prophétie auto-réalisatrice ? Ce qui est clair c'est que pour l'instant aucun projet (Microsoft Windows, GNU/Linux, Firefox …) n'a cassé ces lois. Structure trop complexe et trop importante, contraintes économiques, intérêt à réutiliser du code déjà développé … Ou peut être que la loi de Moore n'est pas prête à être abandonnée car elle est une excuse pour continuer dans cette voie.

Les constats comme celui de GreenIT.fr renforcent cette idée et permettent aux éditeurs de logiciel de s'affranchir de toute contrainte matérielle. D'autant plus que le « nuage » est là pour nous sauver… Un corollaire "vert" ou "durable" à la loi de Moore et pour le domaine logiciel n'est pas encore pour demain. Ce livre tente d'apporter une brique à cet édifice et vous invite à participer à la construction !

---

[9] http://plg.uwaterloo.ca/~migod/846/2008-Winter/project/YanShahab-ProjectReport-FirefoxEvolution.pdf

Le bloatware

Qu'est-ce que réellement le bloatware ? On l'appelle aussi l'obésiciel. C'est de façon générique un phénomène qui se traduit par une baisse de performance des systèmes informatiques entre des versions successives du logiciel. Le logiciel consomme plus de mémoire, est plus lent, ralentit le démarrage du PC… Pour une même fonctionnalité on perd en performance. Ce phénomène peut provenir de deux causes distinctes :

1. des logiciels non nécessaires polluent un système par leur prolifération, nous l'appellerons le bloatware de parasitage,

2. le logiciel ou le système d'exploitation devient plus lent, nous l'appellerons le bloatware entropique.

Le bloatware de parasitage se retrouve dans les PC par la présence d'une multitude de logiciels qui sont lancés dès le démarrage du système. Ces logiciels n'apportent pas de valeur ajoutée ou ne servent à rien. On peut lister quelques exemples :

1. Logiciels constructeurs de gestion du PC (Dell, Sony…). Ces logiciels n'apportent pas grand-chose pour l'utilisateur car les fonctionnalités offertes sont déjà présentes dans les systèmes d'exploitation.

2. Logiciels pré-installés en version d'évaluation (tels des logiciels antivirus). Ces logiciels, une fois la période d'évaluation finie, reste généralement actifs mais non utilisable.

3. Logiciels utilitaires (barre d'outils…). Ces utilitaires, installés de base ou en complément d'autre logiciel, sont actifs tout le temps mais peu utiles car les fonctionnalités sont déjà offertes ailleurs. Par exemple les barres de recherche Google ou Yahoo ne sont pas nécessaires car disponibles

directement dans tous les navigateurs internet.

Le bloatware entropique est le phénomène que l'on retrouve le plus dans nos logiciels. Les logiciels et systèmes d'exploitation sont de plus en plus lents, prennent de plus en plus de place. L'implémentation de nouvelles fonctionnalités est en grande partie la cause de cette entropie. Nous avons vu précédemment la formulation théorique de ce phénomène dans la loi de Lehman, regardons donc cela de façon plus pragmatique.

Les fonctionnalités sont implémentées, au fur et à mesure, dans des versions successives d'un logiciel. Cependant les évolutions sont toujours intégrées sur la même base, le logiciel devient donc de plus en plus complexe et lourd. Une multiplicité de fonctionnalités sont alors embarqué et ont obligatoirement un effet sur l'environnement. Or elles ne sont pas toutes nécessaires pour tous les utilisateurs. La loi de Pareto permet de comprendre cela : environ 20% des fonctionnalités sont utilisés par 80% des utilisateurs, les 80% restantes ne sont donc pas utilisées par la plupart des utilisateurs.

Voici une étude sur l'évolution du système Microsoft Windows entre la version 95 et la version Seven[10] :

[10] http://www.ecoinfo.cnrs.fr/spip.php?article211



Figure 3 - Evolution de la mémoire vive

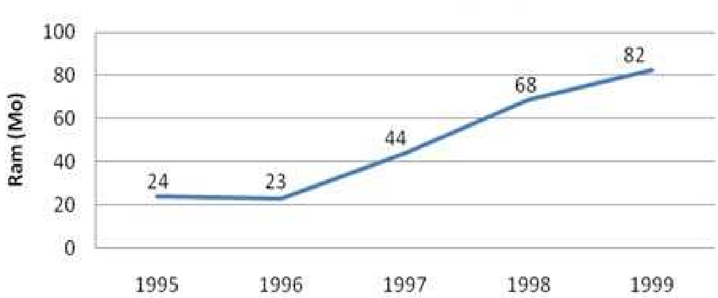


Figure 4 - Evolution des configurations RAM



Figure 5 - Evolution des configurations ROM

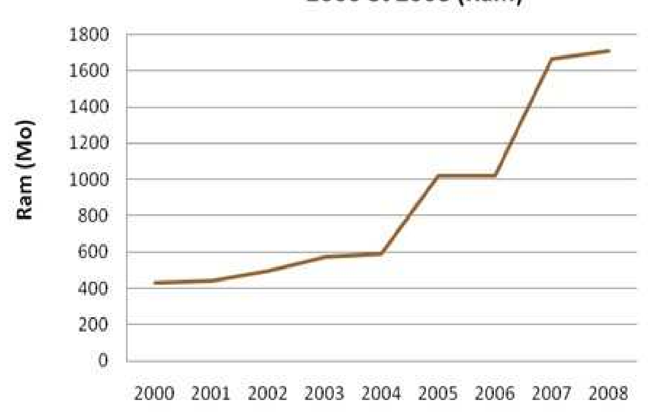


Figure 6 - Evolution des configurations RAM

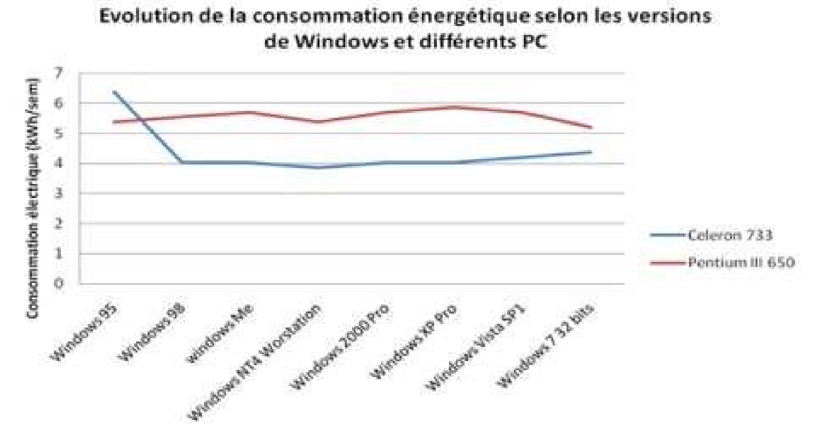
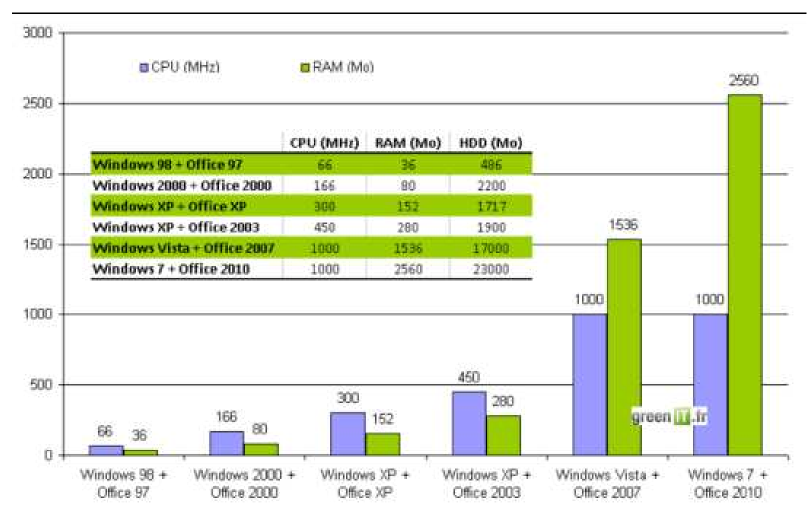


Figure 7 - Evolution des consommations énergétiques

On voit dans ces courbes qu'entre la version 95 et la version Seven, les besoins ont augmenté jusqu'à plus de 150 fois. Or entre ces versions, la fonctionnalité pure du système (et surtout le besoin de l'utilisateur n'ont pas changé) : faire des tâches bureautiques et aller sur internet. La consommation (pour un même matériel) n'a pas évolué mais cela est annulé par la demande croissante en ressources matérielles. Cette demande est réellement négative pour l'environnement car la partie la plus impactante est la fabrication et la fin de vie des équipements, plutôt que leur 'utilisation. Et ce d'autant plus que l'obésité des logiciels se cumule comme le démontre une étude menée en 2010 par GreenIT.fr



	CPU (MHz)	RAM (Mo)	HDD (Mo)
Windows 98 + Office 97	66	36	486
Windows 2000 + Office 2000	166	80	2200
Windows XP + Office XP	300	152	1717
Windows XP + Office 2003	450	280	1900
Windows Vista + Office 2007	1000	1536	17000
Windows 7 + Office 2010	1000	2560	23000

Figure 8 – Evolution des besoins en ressources matérielles du couple Microsoft Windows - Office

Le bloatware est donc un phénomène systématique sur le marché actuel du logiciel. Il est une des raisons de la demande croissante en matériel, de l'augmentation de la consommation d'énergie et du changement régulier de matériel par l'utilisateur.

ASPECTS ECONOMIQUES

Inefficacité du processus logiciel

L'anarchie applicative coûte 6 % du budget informatique

Une récente étude conduite par Coleman Parkes Research pour le compte d'HP estime que 5,8 % des budgets informatiques sont consacrés à des applications sous-utilisées qui accaparent des budgets et des ressources humaines qui pourraient être consacrées à l'innovation. Les DSI sondés estiment que près de 15 % des applications métiers sont sous-utilisées et qu'elles n'apportent pas de réelle valeur. Un tiers des sondés estime même à plus de 20 % la part des applications sous-utilisées dans leur organisation. Maintenir et supporter ces applications qui n'apportent que peu de valeur aux métiers – sinon aucune – conduit à une augmentation inutile des coûts de maintenance. À l'échelle européenne, ce gâchis s'élève chaque année à 16 milliards de dollars.

Les entreprises françaises sont les plus concernées avec 83 % des DSI qui reconnaissent le problème, contre 58 % pour leurs homologues britanniques et 74 % en moyenne en Europe. Les fournisseurs d'énergie et les opérateurs télécoms concentrent ces dysfonctionnements plus que tout autre secteur d'activité.[11][12]

---

[11] http://www.1e.com/mediacenter/Description.aspx?ID=95, mars 2011

[12] HP 2010 Application Management Survey, Coleman Parkes Research Ltd., octobre 2010

Logiciels inutilisés : 27 milliards de gâchés chaque année

Combien coûtent les logiciels que les entreprises achètent sans jamais les utiliser ? Plus de 27 milliards de dollars par an, uniquement aux Etats-Unis et en Angleterre. C'est ce que révèle une enquête menée par Opinion Matters auprès de 500 responsables informatiques dans ces deux pays.

70 % des entreprises admettent acheter plus de licences qu'elles n'en utilisent. 80 % des décideurs estiment que plusieurs logiciels sont installés sur les postes de travail mais ne sont jamais utilisés. En moyenne, 10 % des logiciels achetés et installés ne sont pas utilisés. Au coût de licence - 415 dollars en moyenne - il faut ajouter les coûts de mise à jour et de maintenance. Soit un gâchis compris entre 145 et 155 dollars par an et par utilisateur, uniquement pour la maintenance.

Cette situation est due à l'absence ou la mauvaise gestion des actifs logiciels. Plus de 50 % des entreprises utilisent un tableur pour suivre leurs licences et le déploiement des logiciels, 10 % assurent le suivi sur papier, et plus de 12 % ne gère pas du tout leurs actifs logiciels. La majorité des responsables informatiques sont conscients de l'inefficacité des outils qu'ils ont mis en place. Mais ils considèrent que les logiciels de gestion des actifs logiciels sont trop complexes à utiliser.

20 % des applications d'entreprise sont redondantes

Face à la compression des budgets informatiques, les informatiques vont être poussées à rationaliser leur système d'information. L'inventaire logiciel est une première étape incontournable.

Dans leur premier rapport annuel Applications Landscape[13] qui mesure l'état des systèmes d'information des entreprises, HP et Capgemini constatent que « des millions d'applications sont obsolètes, inutiles, et n'apportent plus aucune valeur pour leur activité ». 85 % des responsables informatiques interrogés estiment que leur portefeuille d'applications doit être rationalisé et 60 % qu'ils gèrent « plus » ou « beaucoup plus » d'applications que nécessaires pour le bon fonctionnement de leur entreprise. Seulement 4 % des DSI estiment que tous leurs systèmes informatiques sont indispensables.

Selon l'entreprise, entre 10 % et 50 % des applications pourraient être supprimés sans nuire à l'activité. La gestion des applications est d'autant plus importante que certaines en gèrent jusqu'à 10 000. En actualisant leur portefeuille d'applications, les entreprises peuvent dégager des budgets supplémentaires pour soutenir l'innovation. Il est donc urgent de rationaliser les portefeuilles applicatifs, notamment en supprimant les logiciels qui consomment inutilement des ressources.

« Malgré les économies substantielles et les gains significatifs d'efficacité et d'agilité qui peuvent en découler, l'archivage des données et le retrait des applications ne figurent toujours pas parmi les premières priorités » résume Ron Tolido, CTO de Capgemini. Le rapport identifie les freins à une gestion efficace du portefeuille applicatif. Si le retrait d'applications n'est pas considéré comme une priorité absolue, d'autres facteurs - coût de leur retrait, absence de retour sur investissement immédiat, résistance culturelle au changement, pénurie de développeurs qualifiés pour la migration des données des applications supprimées, etc. – freinent les DSI dans leur grand ménage de printemps.

---

13 http://www.capgemini.com/discover/pdf/dilemma_5/Application_Landscape_Report_2011_Edition.pdf

Au-delà des applications, plus de 6 entreprises sur 10 conservent toutes les données au-delà de leur date d'expiration, « juste au cas où... ». Or, avec un accroissement du volume des données de + 5 % par mois, le stockage de ces données devient un enjeu économique majeur pour les DSI.

Question à se poser sur le modèle économique

La prise en compte du modèle économique des logiciels est importante pour que le monde du développement logiciel respecte les 3 piliers du développement durable. Pour n'importe quel modèle économique, il faut se poser certaines questions.

- ✓ Le modèle répond-t-il bien aux besoins des utilisateurs et offre-t-il des choix alternatifs ? Un modèle qui ne permettrait pas un gain pour l'utilisateur et pour l'éditeur/développeur. Le modèle Microsoft avec Windows peut par exemple être discuté. En effet, la vente obligatoire de Microsoft Windows dans des PC préinstallés ne va pas dans le sens d'une équité client / fournisseur. C'est une des raisons principales des critiques (et aussi des attaques judiciaires) à l'encontre de Microsoft. Le besoin de l'utilisateur est uniquement d'avoir à disposition un PC opérationnel sur lequel il puisse faire de la bureautique, de l'internet… Verrouiller l'utilisation de Microsoft Windows ne répond pas à ce besoin.

- ✓ Question annexe à la précédente : Le modèle économique ne force-t-il pas l'utilisateur à l'achat ? Certaines pratiques consistent à forcer l'utilisateur à acheter des versions successives des logiciels. Sans cet achat, l'utilisateur ne peut plus (ou mal) utiliser les fichiers qui étaient pris en compte par les versions précédentes. De nombreux exemples de montées de version de certains logiciels peuvent malheureusement être donnés.

- ✓ Le modèle économique est-il viable pour l'entreprise ? Le modèle doit permettre à la société d'avoir un retour sur investissement. Sans cette efficacité, il est probable que tous les investissements ne soient pas pérennes et que le logiciel ne soit donc pas durable. Les chiffres donnés dans le paragraphe précédent montrent que le processus logiciel n'est pas forcément économiquement viable. Comment serait

considérée une usine si 40 % des coûts de production était gaspillé ? Il faut avoir la même réflexion sur le monde du logiciel

- ✓ Le modèle permet-il la production d'un logiciel respectant l'environnement ? Est-il fiable et ergonomique pour l'utilisateur ? Sans ce prérequis il est certain que le logiciel sera abandonné par l'utilisateur ou que l'utilisateur aura du mal à appréhender l'outil. Sans une telle ergonomie du logiciel, les impacts peuvent être nombreux : perte de performance, effet rebond négatif (consommation…), non utilisation de toutes les fonctionnalités du logiciel. La réponse à cette question est très liée à la question précédente : un modèle économique viable permettra d'atteindre indirectement le but d'un logiciel.

Ces questions paraissent anodines pour un développement logiciel mais sont primordiales à appréhender pour les acteurs du développement logiciel.

ASPECT SOCIAUX

L'aspect social n'est pas à négliger dans l'éco-conception des logiciels. Considéré comme le premier pilier du développement durable, il prend en compte les aspects d'insertion des populations et de santé des populations. Le lien avec le pilier économique concerne l'équité (au sens de solidarité), et le lien avec l'environnement concerne le vivable (au sens de santé et de pollution). Compte tenu de la place prépondérante des TIC et donc du logiciel dans notre société, l'aspect social du logiciel est à étudier. On associe souvent le logiciel comme un facilitateur d'inclusion : connexion et communication grâce aux réseaux sociaux, mise en place de logiciels de prévention des risques sanitaires... Cet aspect est certain mais il ne fait pas partie de l'éco-conception. En effet, il est plutôt associé à l'utilisation du logiciel mais pas à la manière dont est fait le logiciel. L'éco-conception des logiciels va se focaliser sur la manière de rendre le logiciel accessible à tous. De cette manière le logiciel répondra au besoin de tous et n'écartera pas certaines personnes. Sur l'aspect lien avec l'environnement, le logiciel évitera de créer des pollutions (ondes radio par exemple)

L'exclusion des populations du numérique est un sujet dont on parle régulièrement via le terme de fracture numérique. Le rapport Attali (Rapport de la Commission pour la libération de la croissance française)[14] propose une définition de la fracture numérique en France et fixe plusieurs objectifs pour réduire la fracture numérique :

- ✓ Démocratiser le numérique en accélérant le déploiement des infrastructures, en garantissant une couverture numérique optimale en 2011, et en réalisant l'accès pour

---

14 http://lesrapports.ladocumentationfrancaise.fr/BRP/084000041/0000.pdf

tous au très haut débit en 2016. L'objectif est d'éviter que certaines personnes, même si elles maîtrisent le numérique, n'aient pas la possibilité d'accéder à internet.

✓ Réduire les fractures numériques en facilitant l'accès de tous au réseau numérique par :

  - L'accélération du taux d'équipement dans les PME/TPE et les foyers
  - Le renforcement des Espaces Publiques Numériques (EPN)
  - Développer l'apprentissage des TICs à l'école
  - Vérifier l'apprentissage lors du passage en 5ème
  - Développer l'apprentissage pour tous à domicile

Il est intéressant de voir dans ce rapport que les premières raisons de cette fracture sont la complexité d'utilisation (29%) et l'absence d'utilité des fonctionnalités offertes par les TICs pour la vie commune (20%).

Plusieurs actions ont vu le jour ces dernière années pour traiter de cette fracture comme l'acquisition de PC à taux préférentiel. Des actions de mise à disposition d'environnement logiciel libre ont aussi été réalisées et ont permis à de nombreuse personnes d'accéder au numérique.

Cependant cette fracture numérique n'est pas si simple à traiter car il ne suffit pas d'avoir du matériel et les logiciels pour pouvoir les utiliser. La complexité des logiciels crée dans certains cas une barrière qui ne permet pas à tous les utilisateurs d'accéder pleinement les fonctionnalité. Il se crée alors une fracture entre les utilisateurs expérimentés (passionnés de technologies, informaticiens, personnes

formées...) et des non-initiés. Cette fracture est encore plus forte avec l'avènement des technologies internet. Le géographe Mark Graham[15] définit cette fracture dans le « village global » (tel que certains parlent d'internet) comme une « cyber-fracture » (en opposition à la fracture matérielle).

Formulée d'une autre manière par Elie Michel dans « Le fossé numérique. L'Internet, facteur de nouvelles inégalités ? ».[16] :

« D'une manière générale, le fossé numérique peut être défini comme une inégalité face aux possibilités d'accéder et de contribuer à l'information, à la connaissance et aux réseaux, ainsi que de bénéficier des capacités majeures de développement offertes par les TIC. Ces éléments sont quelques-uns des plus visibles du fossé numérique, qui se traduit en réalité par une combinaison de facteurs socio-économiques plus vastes, en particulier l'insuffisance des infrastructures, le coût élevé de l'accès, l'absence de formation adéquate, le manque de création locale de contenus et la capacité inégale de tirer parti, aux niveaux économique et social, d'activités à forte intensité d'information. »

Cette problématique d'exclusion créée par le logiciel doit être prise en compte dès la conception du logiciel. Parmi les bonnes pratiques, le concepteur devra intégrer dans le besoin de répondre au plus grand nombre. Cela se traduira dans l'interface d'utilisation, dans l'ergonomie, dans des aides à l'utilisation... L'utilisation de terme technique ou d'anglicisme par exemple sera à proscrire. L'accessibilité des logiciels et site web aux personnes handicapés et âgées est aussi à traiter dans cette optique. Selon les Nations Unies, il y avait, en 2005, 600 millions de personnes en situation de handicap dans le

15 « Time machines and virtual portals : The spatialities of the digital divide »,*Progress in Development Studies*, vol. 11, n° 3, pp. 211 -227, 2011

16 Problèmes politiques et sociaux, La Documentation française, n° 861, août 2001, p. 32

monde. Intégrer l'accessibilité dans les logiciels est donc une priorité importante pour les développeurs.

Ces aspects seront traités dans le chapitre « Intégration du pilier social » p.82.

Pour aller plus loin

Institut de l'accessibilité numérique http://www.accessibilite-numerique.org/

ETAT DE L'ART DES ACTIONS EN COURS

Des actions dans le domaine de l'éco-conception des logiciels sont en cours. Voici une liste non exhaustive impliquant de nombreux acteurs (dont le Green Code Lab pour certaines).

Projet Code Vert

Un consortium d'acteurs regroupant les sociétés KaliTerre et TOCEA et l'école d'ingénieur ICAM de Nantes ont lancé le projet « Code Vert » dont l'objectif est de développer un logiciel qui permettra de qualifier la qualité « Durable » d'un développement informatique. Cet outil utilisera un référentiel de bonnes pratiques de conception logicielle plus respectueuses de l'équilibre environnemental depuis sa phase de création jusque sa phase déchet. Ce référentiel sera issu d'une part des bonnes pratiques de ce livre et permettra d'autre part d'alimenter concrètement celui-ci.

Ce projet est labellisé par le Pôle Images et Réseaux basé à Rennes et démarrera sa phase opérationnelle début 2012. L'outil accompagnera une démarche de conseil « outillé » pour lequel une partie du référentiel développé et validé dans le cadre du projet s'appliquera directement au code (phase de développement). L'outil utilisé sera donc capable de découper un code développé (fonction de parsing), puis de lui appliquer des règles « statiques » de bonne programmation (moteur de règles). Au-delà de la notation accordée au programme (outil de scoring), l'objectif est d'apporter des corrections et/ou des exemples pour mettre en valeur un code « propre » et faire progresser les développeurs.

Groupe de réflexions AFNOR et Syntec

Dans le cadre des groupes de travail Green IT de l'AFNOR et du Syntec (deux groupes indépendants) autour des TIC durables, des questions se sont posées sur l'impact du logiciel. Des groupes de travail sont donc en cours pour traiter le sujet de l'éco-conception des logiciels. Des travaux que le Green Code Lab suit bien-sûr !

Green Software Engineering

Le projet Green Software Engineering est un projet collaboratif allemand qui a pour but d'offrir aux développeurs des outils pour des logiciels plus vert. Le projet vient de publier des résultats intéressants pour le design de sites web.

Le projet tente de définir une cycle de vie du site web calqué sur l'analyse du cycle de vie (ACV). A chaque phase est associé un rôle.

Green Challenge

Faisant suite au Green Challenge organisé par l'USI et GreenIT.fr (voir Chapitre « Applications Web » p.244), le Green Code Lab prépare un challenge de développement qui permettrait de mettre en compétition des développeurs afin d'identifier des green design patterns.



Conception

La conception d'un logiciel respectueux de l'environnement doit suivre les mêmes principes que n'importe quel produit éco-conçu. La compréhension du principe d'analyse de cycle de vie est donc primordiale. De plus, une réflexion profonde sur le processus de développement doit être faite pour couvrir un des piliers du développement durable : l'économie. Les nouvelles pratiques comme l'agilité répondent-elles à ces attentes ?

ACV ENVIRONNEMENTALE ET SOCIALE

Comprendre l'éco-conception d'un produit ou d'un service

Introduction

Pour traiter de l'éco-conception d'un produit, il faut tout d'abord s'attacher à procéder à une analyse de son cycle de vie et d'en déduire une cartographie de ses impacts sur les indicateurs d'impacts environnementaux.

L'ACV (Analyse Cycle de Vie) d'un produit ou d'un service consiste à inventorier les flux de matières et d'énergies entrants et sortants à chaque étape du cycle de vie d'un produit. A partir de ces données, on procède à une évaluation des impacts environnementaux.

La série des normes ISO 14040, parue à partir de 1997, décrit la méthodologie et la déontologie que doivent suivre les études ACV.

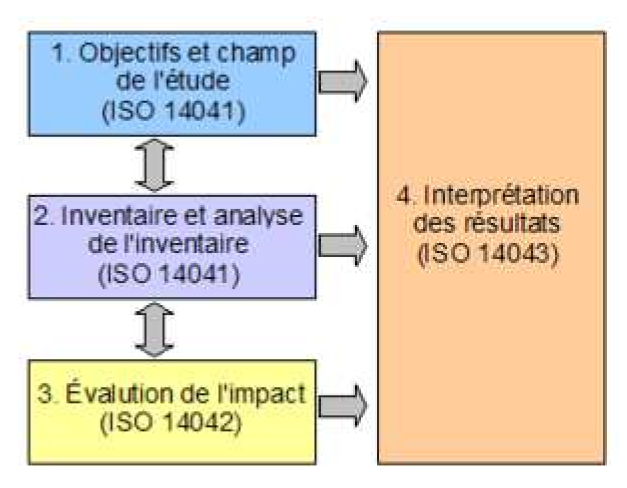


Figure 8 - ACV et ISO (Source wikipédia)

Les impacts environnementaux du berceau à la tombe

Ces indicateurs doivent être mesurés tout au long du cycle de vie du produit. Les principales phases qu'on s'accorde à étudier dans une ACV sont :

- ✓ l'extraction des matières premières
- ✓ le processus de fabrication et de conditionnement
- ✓ le processus logistique et d'acheminement jusqu'au point de distribution
- ✓ la distribution
- ✓ la phase d'usage du produit
- ✓ la fin de vie (recyclage, déchet)

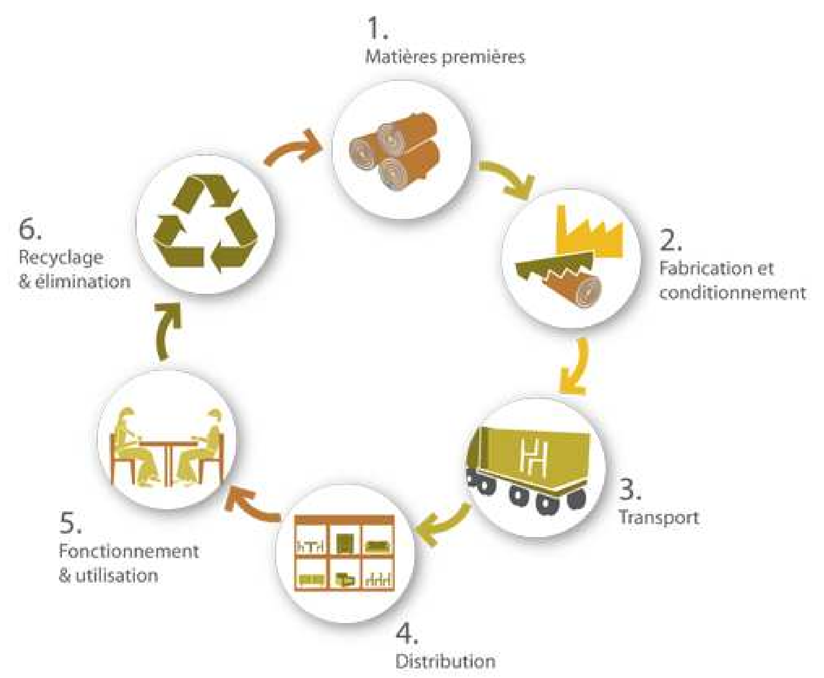


Figure 9 - cycle de vie d'un produit (Source avnir.org)

Les produits et services concernés n'impactent pas les différentes phases de la même manière :

Quelques exemples :

Véhicules 4 roues	Phase d'utilisation : 7 fois plus que pour sa partie fabrication au moins sur l'indicateur émission de GES (Source GNESG.com)
Ordinateur	Phase de fabrication : la fabrication d'un ordinateur en Chine émet 70 à 100 fois plus d'équivalent CO2 que un an d'utilisation en France. (Source greenIT.fr)
Imprimante	Phase d'utilisation : 67 % des émissions de GES se concentrent lors de l'utilisation (papier, toner, kWh électriques, etc.). – source : Lexmark

Figure 10 - Exemple d'impact sur le cycle de vie

Une analyse environnementale multicritères

Il existe un certain nombre de critères qui permettent de qualifier et cartographier l'impact environnemental :

- ✓ le changement climatiques / réchauffement climatique,
- ✓ la destruction de l'ozone stratosphérique,
- ✓ l'acidification,
- ✓ l'eutrophisation,
- ✓ la formation d'agents photo-oxydants (smog),

- ✓ l’atteinte des ressources abiotiques,
- ✓ l’atteinte des ressources biotiques,
- ✓ l’utilisation des terres,
- ✓ l’impact éco-toxicologique,
- ✓ l’impact toxicologique (chez l'humain).
- ✓ l'épuisement des ressources,
- ✓ l'impact sur la santé humaine,
- ✓ les impacts écologiques

Les critères les plus couramment retenus sont l'effet de serre, l'acidification, l'eutrophisation, l'épuisement des ressources naturelles. On retient également en général comme indicateurs la consommation d'énergie et la quantité de déchets générés.

En comparaison d’une analyse de type bilan Carbone® de l’ADEME (Agence de l’Environnement et de la Maîtrise de l’Energie), l’approche ACV permet une analyse multicritère qui permet d’éviter des transferts de pollution au niveau environnemental : par exemple agir sur le Carbone mais ne pas se soucier de la consommation d’eau (cas d’une tasse / gobelet en plastique pour boire un café)

L’éco-conception environnementale, sociale et économique

L’éco-conception traditionnelle basée sur l’ACV d’un produit ou d’un service se borne à mesurer les impacts environnementaux mais il est primordial que cela ne se traduise pas par un déséquilibre sur les axes économique et social :

- ✓ mon produit est-il viable ?
- ✓ respecte-t-il des principes équitables dans sa phase de production, de commercialisation ?
- ✓ quels sont les impacts sur le développement humain ?
- ✓ Quels sont les impacts sur le développement social ?
- ✓ mon produit est-il discriminant ou accessible au plus grand nombre ?

Pour aller plus loin

Site de l'ADEME
http://www2.ademe.fr/servlet/KBaseShow?sort=-1&cid=96&m=3&catid=12922

L'ACV du logiciel

Introduction

L'ACV d'un logiciel n'est pas des plus faciles à appréhender. En effet, ce produit ne génère pas de déchets « Visibles », n'a pas d'obsolescence intrinsèque et provient de sources de production liées au matériel et d'hommes qui ont conçu et programmé le code.

Idées d'action Green Code Lab :

Mettre en place une méthodologie d'ACV logicielle

Aucun ACV de logiciel à notre connaissance n'a encore été publié. Les publications françaises les plus récentes

ont été faites sur l'usage des courriels et factures électroniques (Etude ADEME 2011).

Pour simplifier l'ACV d'un logiciel, on peut le représenter dans 4 phases de son cycle de vie :

- ✓ Fabrication
- ✓ Distribution
- ✓ Utilisation
- ✓ Fin de vie / réutilisation / recyclage

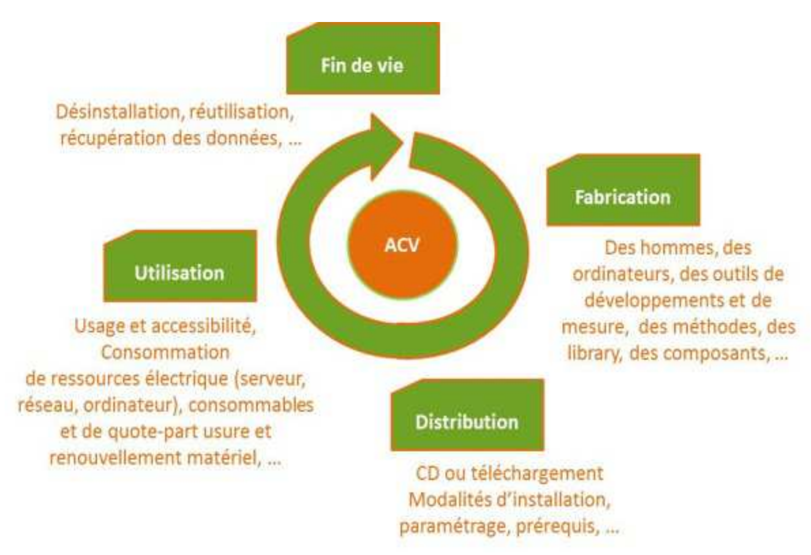


Figure 11 - cycle de vie d'un logiciel (Source kaliterre.fr)

La fabrication

La fabrication d'un logiciel nécessite des moyens humains, techniques (ordinateurs, serveurs, réseaux, périphériques informatiques, …), des moyens de déplacements (sur site de travail, réunion extérieures) et d'hébergement /communication (bâtiment, hébergement, téléphonie).

Les impacts environnementaux sont dus essentiellement aux déplacements dans cette phase. Néanmoins, on peut considérer qu'ils ne sont pas différents des impacts d'une entreprise de service faiblement matérialisés. Dans cette phase, on s'attachera surtout à :

- ✓ Réduire les déplacements sans détériorer l'efficacité et le lien social d'équipe et besoin d'appartenance par des moyens simples et reconnus que sont les vidéo et visioconférences, les audioconférences, le télétravail partiel, les espaces collaboratifs de travail en accès distants, …

- ✓ Prolonger la durée de vie et partager des infrastructures techniques de développement

- ✓ Favoriser le travail de gestion de configuration de développement (Subversion, GIT, …), les espaces de stockages partagés.

- ✓ Travailler dans un cadre agréable et professionnel qui tient compte des règles de sécurité et de santé dans le cadre d'horaires conventionnels

- ✓ Privilégier les économies d'énergies sur les infrastructures techniques de développement (veille /extinction serveurs, veille /extinction ordinateurs, périphériques, ...) et sur les consommables (papier, encre, fournitures diverses).

- ✓ Favoriser la non-discrimination, quelle qu'en soit sa forme, interne ou en sous-traitance des développements.

La Distribution

La distribution d'un logiciel nécessite des moyens techniques (ordinateurs, serveurs, réseaux, périphériques informatiques, …), des moyens humains et éventuellement des moyens d'affrètement depuis un site de production, vers un entrepôt de distribution, vers un circuit de ventes (GMS, magasins spécialisés) d'un CD et son livret d'installation.

Ce dernier type de distribution matérialisé n'est pas différent d'un produit de consommation non périssable. Le mode de distribution non matérialisé par une installation via téléchargement internet est l'alternative permet un moindre impact environnemental.

L'ACV sur cette partie s'attachera donc à mesurer l'impact :

- ✓ de la distribution jusqu'au point de vente
- ✓ des conditions de vente
- ✓ de l'acheminement du programme jusque l'ordinateur de l'utilisateur final
- ✓ des conditions et pratiques d'installations

Les impacts environnementaux sont dus essentiellement aux transports des produits dans cette phase. Néanmoins, on peut considérer qu'ils ne sont pas différents des impacts d'une entreprise de bien de consommation. Dans cette phase, on s'attachera surtout à :

- ✓ Privilégier une distribution des logiciels non matérialisés par téléchargement / installation locale sur le poste de travail de

l'utilisateur ou l'accès à un service en ligne,

- ✓ Dans le cas d'installation non matérialisées, mettre en œuvre une distribution par usage ou service du logiciel qui permet une utilisation par niveau / profil pour permettre un gain de disque, un gain réseau et une facilité d'usage

- ✓ Eviter les pratiques d'installation trop gourmandes en disque ou non sollicitées

- ✓ Eviter des prérequis surdimensionnés qui impliquent une décision de choix de remplacement d'un ordinateur.

- ✓ Privilégier les mises à jour régulières et incrémentales pour limiter les flux réseaux et le non –traitement des failles de sécurité.

- ✓ Intégrer des pratiques commerciales claires sur la base de l'usage du logiciel.

- ✓ Désinstaller les données non nécessaires à l'issue de l'installation

- ✓ Respect des données personnelles

L'utilisation

Le logiciel ayant pour objectif d'être diffusé au plus grand nombre (hors cas de logiciel très spécifique de calculs scientifiques par exemple), son impact est décuplé si la prise en compte très en amont de bonnes pratiques de conception ne sont pas intégrées.

D'un point de vue environnemental, 3 impacts majeurs sont à prendre en compte :

- ✓ L'énergie consommée « instantanée » lors de l'utilisation du

logiciel par le matériel : énergie de l'ordinateur (CPU, disque, RAM, périphériques), du ou des serveurs, du réseau.

- ✓ L'énergie et les matériaux consommés pour produire le matériel (ordinateurs, serveurs, réseaux, périphériques) qui permet d'exécuter le logiciel avec sa quote-part d'utilisation et d'usure sur la base d'un nombre d'années d'utilisation. Ce facteur est renforcé ces dernières années par le remplacement prématuré des matériels (tous les 3 ans).

- ✓ L'utilisation de consommables par le logiciel (papier, encre…).

D'un point de vue humain et social, les impacts principaux que nous retiendrons, sont :

- ✓ L'accessibilité au plus grand nombre des fonctionnalités et contenus

- ✓ L'orientation « Usage » : le logiciel doit s'adapter à l'usage qui en est fait à croiser avec les profils de ceux qui l'utilisent

- ✓ La qualité et sécurité du produit qui évite toute perte de temps, de données, de biens financiers, …

- ✓ Le respect des données personnelles

D'un point de vue économique, les impacts principaux que nous retiendrons, sont :

- ✓ Un modèle équitable commercialement intégrant une transparence des coûts d'utilisation et des coûts de maintenance (coûts cachés par exemple avec par exemple la maintenance offerte la première année mais un coût non négligeable les années suivantes)…

- ✓ Dans cette phase, on s'attachera surtout à travailler la conception et la mesure dans les phases de réalisation pour éviter une surconsommation dans l'utilisation :

    - o Etudier les choix d'architecture du logiciel et penser à une décomposition des fonctionnalités par service

    - o Etudier les langages, librairies, composants, module graphiques et s'attacher à réutiliser quand nécessaire et à optimiser (quitte à re-développer) des fonctions majeurs et consommatrices.

    - o Faire le lien avec le matériel pour piloter au mieux

    - o Mettre en place des outils de mesure de la qualité, sécurité et de la consommation pour favoriser l'amélioration

    - o Optimiser la volumétrie des données nécessaires (non redondance, optimisation des formats, fonctionnalités de purge – historisation, …)

Fin de vie / réutilisation / recyclage

C'est de loin la phase la plus difficile à appréhender en terme environnemental, social et économique pour un logiciel. Le logiciel ne génère pas de déchet physique. Néanmoins, ce produit n'échappe à la règle de durabilité du produit : plus on prolonge sa durée de vie par des adaptations technologiques et fonctionnelles nécessaires, plus on limite les impacts.

L'ACV sur cette partie s'attachera à mesurer également l'impact :

- ✓ De la fin de vie du logiciel en ce qui concerne sa désinstallation

- ✓ De la fin de vie des données du logiciel (recyclage, réutilisation, anonymisation) ou de l'accès à un service (désinscription)

Dans cette phase, on s'attachera surtout à travailler :

- ✓ La pérennisation du code par sa documentation et sa capacité à évoluer

- ✓ Des fonctionnalités de désinstallation « propres » sur le poste local ou la désinscription à un service en ligne facilitée

- ✓ Une facilité de récupération des données générées dans la cadre de l'utilisation de l'outil soit par un outil de récupération sur un format pivot ou format standard du marché.

- ✓ La possibilité d'interrompre facilement l'utilisation du logiciel ou d'un service en ligne sans contrepartie financière dissuasive.

Effectuer une ACV

Calculer l'impact de la fabrication

Il existe quelques analyses du cycle de vie de logiciel. On peut par exemple citer les analyses ADEME sur les requêtes web ou l'envoie d'un courriel[17]. Ces études se focalisent en effet sur un ensemble intégrant logiciel et matériel. On appelle cela le périmètre fonctionnel : C'est le service rendu par le produit et permet de comparer l'impact de deux produits ayant la même unité fonctionnelle. Logiciel et matériel sont étroitement liés,

---

[17] Etude ADEME : http://www.thr34.fr/Fichiers%20echange/Eco_conception/formation%20ecoconception_partie1.pdf.

cette unité fonctionnelle est donc la plus cohérente. En effet, le logiciel sans matériel ne peut pas rendre de service à l'utilisateur, car il lui faut obligatoirement une plate-forme matériel. Cependant, la maîtrise de l'impact du logiciel nécessite une analyse plus fine et donc le besoin d'une unité fonctionnelle du logiciel uniquement ou alors une séparation plus claire des effets du logiciel. La raison du manque de telles analyses provient du fait que le domaine de l'éco-conception des logiciels soit nouveau mais surtout que de nombreuses questions se posent lors de ces études sur l'effet du logiciel sur le matériel. On peut de plus reprocher aux études existantes qu'elles ne prennent pas en compte les phases de fabrication du logiciel. En première approche, on peut estimer, intuitivement, que l’impact de la phase de développement du logiciel est minime par rapport à celui de la phase d’l'utilisation. Compte tenu du fait que les applications sont amenées a être déployées généralement à grande échelle, on peut considérer que leurs utilisations seront beaucoup plus néfastes que leurs productions. Le processus de développement logiciel, ainsi que son utilisation, est cependant complexe pour affirmer cela. On peut à l’inverse citer les logiciels scientifiques qui nécessitent des années homme de développement pour une utilisation par quelques chercheurs.

Comme nous l'avons vu (Chapitre «Comprendre l’éco-conception d’un produit ou d’un service » p.64), la phase de fabrication du logiciel peut être polluante. Cependant, si il est facile de délimiter le périmètre du produit logiciel fini, il est plus complexe de délimiter les impacts de la fabrication. Contrairement à un produit physique, l'inventaire des éléments nécessaires à la fabrication va devoir être fait en prenant certaines hypothèses :

- ✓ Les ressources nécessaires sont principalement des ressources humaines. Or, l'organisation des projets logiciels est souvent matricielle. Certains experts interviennent en effet sur plusieurs projets. Il en va de même pour les ressources de développement. Il est donc assez difficile de répartir l'impact de ces ressources sur les différents

produits logiciels. Cette analyse sera facilitée en utilisant les systèmes d'affectation et de suivi des équipes sur les projets. Plus on connaîtra qui a travaillé sur le produit, plus on arrivera à inventorier les ressources utilisées pour le développement (Voir calcul de TJEM pour exemple « Pilotage énergétique des projets informatiques » p.105), les déplacements, la quote-part d'utilisation des bureaux, …

- ✓ Même si un produit logiciel est commercialisé, l'effort de fabrication n'est pas terminé. Il va en effet se prolonger sous la forme de la maintenance de correction et d'évolution. Donc l'impact de la phase de fabrication va augmenter. Compte tenu que le coût de maintenance peut atteindre 80 % du coût global de fabrication, cet impact n'est pas à négliger. L'analyse du cycle de vie est donc à mettre à jour régulièrement tout au long de la vie du logiciel.

- ✓ Le logiciel évolue sous différentes formes et plus particulièrement sous différentes versions. Certaines montées de version sont mineures car elles n'intègrent que quelques évolutions mais d'autres peuvent être majeures comme par exemple le passage entre les versions du système d'exploitation Microsoft Windows. On ne doit pas dans la plupart des cas considérer les différentes versions comme des produits distincts, et donc ne pas différentier les impacts de fabrication entre les versions ? Ce n'est pas simple car les développements réutilisent souvent des éléments logiciels déjà existants. Deux solutions peuvent être appliquées : Continuité ou rupture. Si il y a continuité forte dans le développement, la nouvelle version est considérée comme une suite et on hérite donc de l'impact des versions antérieures. Si il y a rupture, alors l'impact des versions antérieures sera prise en compte uniquement pour les modules réutilisées.

- ✓ Toutes les parties du logiciel ne sont pas toujours développées par la société : des librairies sont parfois fournies par d'autres sociétés ou d'autres développeurs.

L'impact de ses modules ne peut pas être négligé. Chaque module intégré doit donc être évalué afin d'en analyser le poids sur le logiciel fini. Des hypothèses devront cependant être faites pour pondérer cette analyse en fonction de l'importance de chaque module et faire que cette évaluation soit raisonnable.

Pistes de réflexions

Plus globalement, pour tout le cycle de vie du logiciel, d'autres problématiques inhérentes aux logiciels se posent. Il n'existe pas encore de solutions claires mais quelques pistes permettent de répondre en partie aux questions.

Unité fonctionnelle et points de fonction

La définition de l'unité fonctionnelle est par exemple critique et complexe pour les applications. Elle permet de comparer différents produits entre eux. Or le logiciel intègre une multitude de fonctionnalité. La définition de l'unité fonctionnelle doit prendre en compte cette contrainte. Comment comparer deux logiciels de bureautique compte tenu du nombre de fonctions, d'options et d'interfaces ? On peut trouver des pistes de réflexions dans les points de fonctions utilisés dans le chiffrage des logiciels. Les points de fonction permettent de mesurer la taille du logiciel en quantifiant les fonctionnalités offertes aux utilisateurs en se basant sur des modèles de calcul. Cette métrique est standardisée par l'IFPUG (International Function Point User's Group) et par l'ISO. La méthode inventorie les éléments suivants[18] et affecte des points :

- ✓ ENT (entrée) : fonctions d'entrée de données de l'utilisateur : Créations, modifications, duplications, validations, suppressions

[18] Wikipedia : http://fr.wikipedia.org/wiki/Point_de_Fonction

- ✓ INT (interrogation) : fonctions de consultation des données : Listes, détails, annotations

- ✓ SOR (sortie) : fonctions de restitution des données transformées : Calculs, graphiques, déductions.

La somme des points attribués à tous les composants donne la taille fonctionnelle du logiciel.

- ✓ Portail complet de vente de pièces détachées sur internet : env. 6 000 points ;

- ✓ Logiciel de navigation, type Carminat, Garmin, ... : 1 000 points ;

- ✓ Logiciel de comptabilité, type Ciel Compta : 2 000 points ;

- ✓ Logiciel de gestion des nomenclatures industrielles : 4 000 points ;

- ✓ Logiciel d'analyse de temps et de la paie dans une grande entreprise : 5 000 points.

La définition de l'unité fonctionnelle peut ensuite se baser sur les points de fonction. Analyser l'impact de deux logiciels est alors plus cohérent si l'on compare des applications ayant le même nombre de points de fonction.

Pour aller plus loin

IFPUG : http://www.ifpug.org

Déploiement et analyse de l'impact

Pour un produit physique, on arrive très bien avec l'unité fonctionnelle à identifier la partie de fabrication nécessaire pour le produit fini. Or pour le logiciel, cela est beaucoup plus complexe : La fabrication du logiciel ne va pas en effet donner une unité mais plusieurs unités. Le logiciel va être en effet déployé à plus ou moins grande échelle. Nous pouvons comparer cela à la démultiplication d'un produit physique (qui elle n'est pas possible). Alors comment prendre en compte cette particularité ? Le déploiement permet de mutualiser l'impact de la phase de fabrication sur tous les utilisateurs mais il augmente aussi les impacts de la phase d'utilisation. Donc, deux approches sont possibles (et complémentaires) :

- ✓ On considère le périmètre d'un logiciel unique. L'impact de la phase de fabrication va donc être divisé par le nombre de logiciels déployés (voir Chapitre « Évaluation de l'impact global » p.223).

- ✓ On considère la totalité des logiciels déployés. On ne divise plus l'impact de la phase de fabrication mais on multiplie l'impact de la phase d'utilisation.

La première approche est utile pour fournir des éléments de comparaison intéressants. Par exemple, si l'on veut évaluer l'impact comparatif de l'intégration de plusieurs logiciels, on pourra utiliser ces données. On pourra alors dire concrètement quel impact prendre en compte. L'impact d'un fournisseur sera beaucoup plus important si le logiciel n'a pas été déployé à grande échelle. Cela est normal car on doit assumer l'impact du fournisseur. Cependant cela ne doit pas être une excuse pour que les développeurs s'affranchissent d'intégrer des bonnes pratiques si ils déploient le logiciel à grande échelle. La deuxième approche est donc nécessaire pour analyser l'impact du logiciel vis-à-vis du déploiement. Une amélioration même mineure de l'impact lors de la phase d'utilisation aura alors un effet important sur un logiciel déployé largement.

INTEGRATION DU PILIER SOCIAL

Comme vu dans le chapitre « Aspect Sociaux » p.56, le logiciel doit s'adresser au plus grand nombre de personne et ne doit pas exclure des parties de la population. Des normes d'accessibilité existent mais ne sont malheureusement pas assez appliquées par les développeurs. De plus, l'accessibilité est souvent associée aux sites internet mais pas aux logiciels.

Normes d'accessibilité

Concernant les sites internet, le World Wide Web Consortium (W3C)[19] élabore des normes d'accessibilité pour les sites internet. Le Consortium Daisy établit des normes pour documents électroniques afin qu'ils puissent être rendus accessibles aux personnes handicapées (XML NISO/DAISY Z39.86, notamment) ;

L'ISO fournit des recommandations pour l'amélioration de l'accessibilité de l'équipement, des services des TIC et des logiciels. Elle couvre les aspects liés à la conception de l'équipement et des services pour les personnes présentant un large éventail de capacités physiques, sensorielles et cognitives, y compris les personnes présentant des déficiences temporaires et les personnes âgées. (ISO 9241). La norme ISO 9241-171:2008 se focalise plus particulièrement sur le logiciel (Ergonomie de l'interaction homme-système et lignes directrices relatives à l'accessibilité aux logiciels). On peut noter parmi les principes généraux de la norme :

[19] http://www.w3.org/WAI/

- ✓ L'information doit être perceptible par l'utilisateur (ISO 9241-12, WCAG 2.0 Principle No 1);

- ✓ Les contenus doivent être compréhensibles (ISO 9241-12, -110 and WCAG 2.0 Principle No 3);

- ✓ Les éléments de l'interface doivent être utilisables (ISO 9241-110 and WCAG 2.0 Principle No 2);

- ✓ Le logiciel doit être tolérant aux fautes logicielles (ISO 9241-110 and DFA); ISO/DIS 9241-171)

- ✓ Le logiciel doit être flexible en utilisateur et doit permettre à l'utilisateur de choisir parmi un large choix d'entrées et de sorties alternatives (ISO 9241-110 and DFA)

Plusieurs checklists permettent de vérifier la couverture de nombreux items de l'accessibilité :

- ✓ Voluntary Product Accessibility Template® (VPAT®)[20] de l'information Technology Industry Council

- ✓ Checklist d'accessibilité IBM[21]

Outils et aides pour les développeurs

Afin de faciliter l'intégration de l'accessibilité dans les applications, des API existent dans différentes technologies et permettent de rendre les interfaces graphiques accessibles et interopérables :

20 http://www.itic.org/index.php?submenu=Resources&submenu=Resources&src=gendocs&ref=vpat&category=resources

21 http://www-03.ibm.com/able/guidelines/software/accesssoftware.html

- ✓ ATK pour Gnome[22]
- ✓ API pour Java[23]
- ✓ Iaccessible2[24] pour GNU/Linux

Coté outils, on peut noter l'outils FANGS qui permet au développeur d'émuler les lecteurs d'écran. Les lecteurs d'écrans permettent d'interpréter ce qui est affiché sur l'écran et donc de rendre accessible l'affichage. Le développeur pourra ainsi voir comment seront traitées les pages par les lecteurs d'écran. Ces derniers permettent d'émuler ce que l'utilisateur verra à l'écran.

Afin de vérifier l'implémentation de l'accessibilité, les méthodologies suivantes peuvent être utilisées[25] :

- ✓ Tests automatiques des règles et des guidelines
- ✓ Evaluation par des experts
- ✓ Evaluation en utilisant des modèles ou des simulations
- ✓ Evaluation avec des utilisateurs potentiels
- ✓ Evaluation via les retours des utilisateurs réels

---

22 http://developer.gnome.org/accessibility-devel-guide/stable/index.html.fr

23 http://java.sun.com/javase/technologies/accessibility/docs/jaccess-1.3/doc/core-api.html

24 http://www.linuxfoundation.org/collaborate/workgroups/accessibility/iaccessible2

25 The evaluation of accessibility, usability and user experience - Helen Petrie - University of York, Department of Computer Science http://www.nigelbevan.com/papers/The_evaluation_of_accessibility_usability_and_user_experience.pdf

Concernant les fichiers générés par les logiciels, le développeur fera attention à utiliser des fichiers compatibles avec l'accessibilité. On peut par exemple noter que les générateur dynamiques de PDF sont difficiles à prendre en compte par les outils d'accessibilité.

PROCESSUS DE DEVELOPPEMENT

Processus de développement plus respectueux de l’environnement

Quick And Dirty

La science du développement est assez jeune contrairement à d'autres domaines. Cela amène à des pratiques elles-aussi jeunes et pas toujours vertueuses. Parmi ces pratiques, celle de la programmation rapide (voir précipitée) est intéressante à noter. De nombreux développeurs appliquent en effet des méthodes que l'on peut appeler « quick and dirty » : rapides et sales. Ces solutions permettent de programmer rapidement car l'on ne s'occupe pas d'appliquer les bons design patterns, de réfléchir à l'architecture... Solutions très répandues, elles permettent d'arriver aux résultats voulus rapidement et de corriger des problèmes en cas de crise. L'exemple le plus célèbre est celui du premier système d'exploitation, MS-DOS dont le nom initial signifiait tout simplement: Q-DOS : « Quick&Dirty Operating System ».

Le problème de cette pratique est que cela crée une dette technique. Ce terme a été utilisé la première fois par l'expert Ward Cunningham en 1992 pour expliquer l'implication d'une complexité technique. En comparant cette dette technique à une dette financière, le but était de montrer qu'un code mal conçu requiert le paiement d'intérêts dans un délai plus ou moins long. Quels intérêts ? Un effort de maintenance, un effort d'évolution ou même un travail pour nettoyer le code. Sans parler de la nécessité de changer les ordinateurs (poste de travail notamment) à chaque nouvelle version. Dans tous les cas, cette dette contractée dès l'écriture du code devra être payée par l’entreprise ou le particulier qui utilisent le logiciel.

Si l'on regarde maintenant l'impact environnemental, qu'amène

le quick-and-dirty ? Tout d'abord, avec les pratiques quick-and-dirty aucune réflexion n'est faite sur l'architecture ou sur les bonnes pratiques, donc il n'y aura pas plus de réflexion sur l'implémentation de green patterns (Voir chapitre « Green Patterns » p.163) ou de durabilité du logiciel. Pour ce qui est de la dette technique, l'impact se fera principalement sur les problématiques de maintenance : logiciel qui va devenir instable, code non maintenu et abandonné... Nous verrons tous ces problèmes dans les chapitres suivants. Et le pas est petit pour passer d'une dette technique à une dette environnementale. En effet les pratiques quick and dirty induisent plus de consommation d'énergie, une obsolescence artificiellement accélérée du matériel, et donc son remplacement plus rapide... Mais contrairement à la dette technique qui devra être payée par la société qui développe le logiciel, c'est l'utilisateur et la société qui devront payer la dette économique ! Les intérêts de la dette sont d’autant plus élevés si l’on cherche à minimiser l’impact environnemental. Il faut en effet agir non pas sur le logiciel mais via d’autres axes moins directs. La green IT regorge de telles pratiques pour corriger les mauvaises pratiques de développement : logiciel d’optimisation de la mémoire, logiciel de nettoyage du fichier système, virtualisation, espace de stockage supplémentaire…

Ci-dessous la dette technique vue par scrumalliance.org.

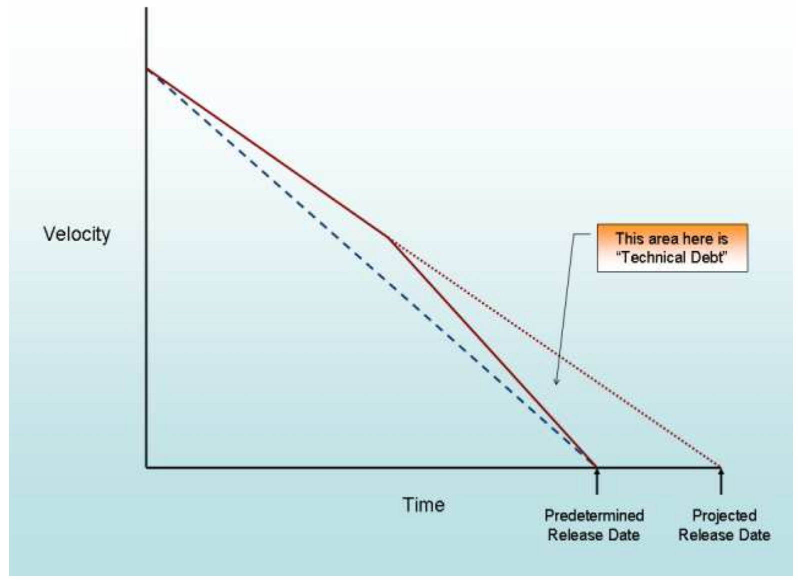


Figure 12 - Dette technique selon scrumalliance.org

De nombreuses discussions dans le monde des développeurs font rage sur l'intérêt du quick and dirty. Pourquoi passer du temps à réfléchir sur des bonnes pratiques alors que le client veut une simple application ? D'autant plus que certains développeurs ont eu la malheureuse expérience d'utiliser des bonnes pratiques sans que le logiciel ne satisfasse le client. Martin Fowler dans son blog introduit certains concepts intéressants sur les différentes natures possibles de la dette technique[26]. La dette peut être considérée comme prudente ou imprudente, délibérée ou non délibérée. Cela permet ainsi de classifier et de bien comprendre cette dette et ses impacts.

[26] http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html

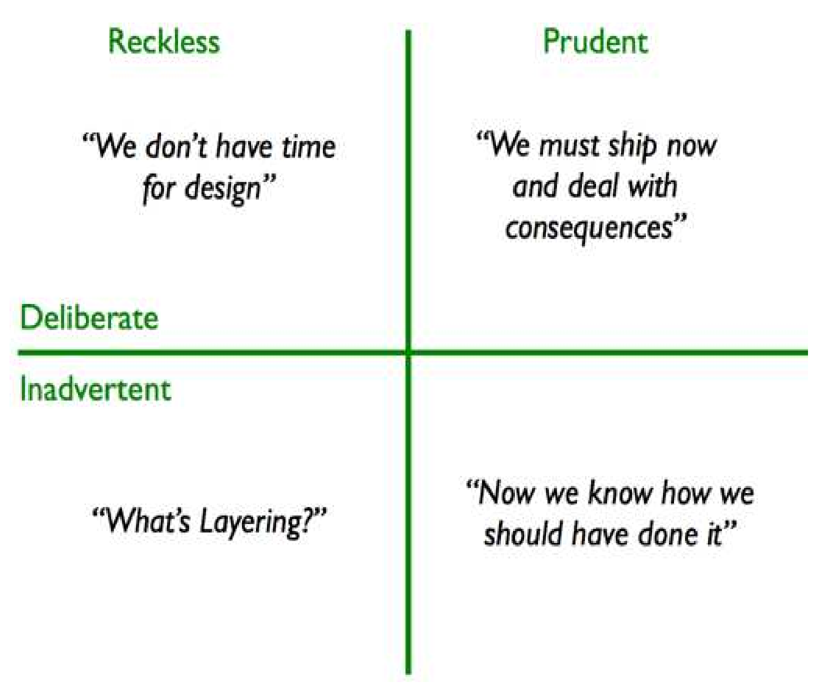


Figure 13 - Classification de la dette technique

Une dette délibérée et imprudente implique que le développeur soit conscient de la non application des bonnes pratiques. La dette n'est donc pas très importante pour lui : l'important ce sont le délai de livraison et la satisfaction immédiate du client. Une dette prudente et délibérée se caractérise par un développeur agissant comme le développeur précédent mais qui sait que la dette devra être payée plus tard. Cette situation est tout à fait correcte car ce serait illusoire de penser que l'on peut tout le temps développer avec des bonnes pratiques face aux différentes pressions. Une dette non délibérée et imprudente arrive quand le développeur ne sait pas qu'il utilise une pratique quick and dirty. Il est débutant, il ne connait pas le langage ou il n'a jamais entendu parler des bonnes pratiques. Quant au dernier quadrant, dette prudente et non délibérée est

la plus compliquée mais la plus courante. Lors de gros projet (plusieurs mois), il n'est pas simple d'appliquer immédiatement des bonnes pratiques. Les développeurs doivent donc avancer à pas de loup. Il est certain pour eux qu'ils sont en train de contracter une dette mais le fait d'avancer prudemment, de surveiller et réfléchir à de meilleures pratiques va permettre de rembourser cette dette dès que possible.

Il ne faut donc pas réfléchir « binairement » en quick and dirty ou bonnes pratiques mais plutôt être conscient que dans tous les cas il y a une dette et qu'il faudra un jour la rembourser. Et cette réflexion doit être faite non seulement sur la dette technique mais aussi environnementale. Par exemple cette dette imprudente et délibérée peut être justifiée si le développeur n'a pas le temps de faire un bon design. Dans ce cas appliquer des bonnes pratiques n'est pas forcément utile car c'est un « petit programme » : la dette technique est dans ce cas justifiée (et encore, cette dette va être importante si votre programme a du succès..), mais qu'en est-il pour la dette environnementale. Qu'arrive-t-il si votre programme pollue la veille du poste sur lequel vous allez l'installer ? Il faut donc bien réfléchir avant de coder non seulement à la dette technique que vous allez devoir rembourser mais aussi à la dette environnementale que vous ou vos utilisateurs devront payer.

Un meilleur processus de développement

Les processus de développement permettent d'améliorer la fabrication d'un code en structurant des phases de développement. De nombreux modèles sont appliqués : cycle en V, RUP… Ils ont tous leurs avantages et leurs contraintes mais ils n'intègrent pas nativement les contraintes environnementales. Si on considère ces contraintes comme d'autres contraintes (réglementaires par exemple) ou comme un besoin, tous ces processus sont compatibles avec le développement durable. En effet l'éco-conception et les green patterns ne sont que des exigences supplémentaires (ou plutôt complémentaires) à celles existantes dans développement

classique (Voir chapitre « Intégration aux processus et normes existantes » p.95). Prenons l'exemple du cycle en V, processus très rependu, ou la partie gauche du V est composé des phases de développement séquencées chronologiquement : expression de besoin, puis architecture, puis conception… et la partie droite du cycle est composée des phases de validation : test unitaire par exemple. On listera les exigences environnementales et les réglementations dans la phase d'expression de besoin. Elles seront ensuite intégrées dans l'architecture et ainsi de suite jusqu'au développement ou les green patterns seront appliquées. De la même manière dans la partie remontante du v on validera que les exigences environnementales sont bien intégrées dans le logiciel. Cette réflexion peut être appliquée à tous les processus de développement.

Certains processus sont cependant plus aptes à intégrer les principes de l'éco-conception. Les processus Agile comme le scrum sont des bons exemples de cette adaptabilité. Ces processus ont pour but d'adapter le produit au besoin client et cela tout au long du processus de développement, de réduire les temps de développement et améliorer la qualité du produit. Afin d'atteindre ses buts, les équipes appliquent des principes que l'on retrouve dans toutes ces méthodologies : séquencement par fenêtre de développement et livraison régulière (sprint), réunion journalière des équipes (daily scrum), réunion en fin de sprint pour analyser les bonnes et mauvaises pratiques (rétrospective…). Cela a pour but d'éviter « l'effet tunnel » qui est la cause de nombreux échecs et dérives de projets informatiques : Le client exprime son besoin, les développeurs acquiescent qu'ils ont compris et partent pour une longue route de plusieurs mois sans contacts avec le client. Arrive la dernière semaine avant la livraison : stress, nuits de travail… pour livrer un projet qui aura sûrement dérivé du besoin du client, qui ne sera pas complet et un peu bogué. La méthodologie agile avec ses principes directeurs évite ce dysfonctionnement.

La méthodologie Agile a donc deux avantages environnementaux :

- ✓ Elle permet d'optimiser le projet de développement : plus d'efficacité donc moins d'impact lors de la fabrication. Cette méthodologie a en effet pour objectif d'éviter certains écueils (effets tunnels, divergences par rapport aux cahiers des charges, retards). Le coût environnemental induit par ce processus n'est pas forcément plus faible initialement mais est mieux maîtrisé.

- ✓ Elle permet d'intégrer facilement les contraintes environnementales. La dette prudente et non délibérée pourra par exemple être réduite dès les premiers sprints. En effet, le fonctionnement par intégration de fonctionnalité progressivement dans chaque itération permet au fur et à mesure de mesurer et de contrôler les écarts par rapport aux contraintes environnementales.

Pour aller plus loin

Livre Choisir l'agilité - Du développement logiciel à la gouvernance – Mathieu Boisvert

French scrum user group (SUG) - http://www.frenchsug.org

On peut nuancer néanmoins cette affirmation pour signifier que des méthodes plus traditionnelles (cycle en V) fournissent des avantages pour la gestion de projets de plus grandes importances dont les besoins sont peu évolutifs et dont le cadre du besoin est parfaitement maîtrisés. Cette méthodes peut être jalonnées de points de contrôle pour s'assurer tout au long du projet les règles d'éco-conception, de durabilité sont bien prises en compte (audit intermédiaires de code, d'accessibilité, …).

Code d'éthique

Il est courant de voir des professions dont l'impact est important sur l'homme ou l'environnement s'appuyer sur des codes ou des serments. L'ACM (Association for computing) et l'IEEE-CS se sont joints pour rédiger un code de l'éthique des développeurs[27]. Ce code a pour but d'offrir aux développeurs des règles leur permettant d'appliquer certains principes en faveur du bien-être de tous :

- ✓ Le public : Les ingénieurs logiciels doivent agir en conformité avec l'intérêt public.
- ✓ Client et l'employeur : Les ingénieurs logiciels doivent agir d'une manière qui est dans le meilleur intérêt de leur client et l'employeur et conforme à l'intérêt public.
- ✓ Produit : Les ingénieurs logiciels doivent s'assurer que leurs produits et modifications connexes sont conformes aux plus hautes normes professionnelles possibles.
- ✓ Jugements : Les ingénieurs logiciels doivent maintenir l'intégrité et l'indépendance de leur jugement professionnel.
- ✓ Gestion : Les responsables de l'ingénierie du logiciel et les dirigeants doivent souscrire et promouvoir une approche éthique de la gestion du développement de logiciels et de la maintenance.
- ✓ Profession : Les ingénieurs logiciels doivent porter la réputation de la profession conformément à l'intérêt public.
- ✓ Collègues : Les ingénieurs logiciels doivent être justes et aider leurs collègues.

[27] http://www.acm.org/about/se-code

- ✓ Soi-même : Les ingénieurs logiciels doivent participer à l'apprentissage tout au long de la pratique de leur profession et promouvoir une approche éthique de la pratique de la profession.

Ces principes paraissent simples mais sont la base de toute démarche de développement durable. Ils rappellent que le logiciel tient une part importante dans le monde actuel et que les développeurs doivent faire preuve de professionnalisme. En effet, l'intérêt public et l'éthique dont font mention ces règles sont cohérentes et complémentaires à celle du développement durable. En effet, le développement durable est bien une conception de l'intérêt public appliquée à la croissance économique en prenant en compte l'environnement et les aspects sociaux.

Ces codes d'éthique sont clairement applicables au monde du développement logiciel. En effet, les développeurs, techniciens et ingénieurs portent une attention croissante à l'aspect social et au développement durable :

- ✓ Ils souhaitent montrer que les évolutions technologiques sont durables

- ✓ La société demande de plus en plus que les techniciens ne se focalisent pas uniquement sur la technique

- ✓ Les développeurs ont conscience qu'ils ont un impact sur la société

- ✓ La demande pour du durable est de plus en plus forte.

On ressent ce mouvement avec l'apparition de plus en plus forte de normes ou de guideline. On ressent cela aussi dans le monde académique : avec de plus en plus de recherche ou de concours étudiants sur le domaine.

INTEGRATION AUX PROCESSUS ET NORMES EXISTANTES

De nombreux processus et normes permettent de piloter les processus informatiques : norme de qualité logicielle comme l'ISO 25 000, méthodologie agile de développement, processus ITIL… Comment s'intègre l'éco-conception dans ces processus ? Tout d'abord il faut noter que ces processus n'ont pas été conçus en prenant en compte les besoins du développement durable. Il est peu probable alors qu'ils répondent nativement à l'éco-conception. Cependant, l'éco-conception n'est qu'un domaine transverse supplémentaire comme la sécurité ou la qualité logicielle. Il est donc normalement faisable d'intégrer l'éco-conception dans les processus comme des contraintes supplémentaires. Vous trouverez dans ce chapitre des exemples d'intégration

Qualité logicielle et ISO 25000

Présentation de la norme

La qualité logicielle (Voir chapitre « 5 Après le développement » p.207) est normalisée par l'ISO 25000. Cette norme définit comment évaluer la qualité du logiciel. Elle définit trois niveaux :

- ✓ Produit logiciel (ISO 25022) : qualité interne du logiciel (efficacité, maintenabilité...). Ces caractéristiques sont internes au logiciel et vont influer (ou pas) sur le système et sur l'utilisation.

- ✓ Produit système (au sens qualité de produit ISO 25023) : qualité externe du logiciel, c'est à dire évaluation du logiciel dans son environnement (PC, mobile, data center...). La qualité interne influe sur la qualité externe.

- ✓ Phase d'utilisation: Caractéristique du logiciel dans un

contexte d'utilisation spécifique (satisfaction,...). Les caractéristiques de qualité des items software product et system product vont permettre d'atteindre les caractéristiques d'utilisation visibles par l'utilisateur.

Produit logiciel

La qualité interne du logiciel peut être définie par des caractéristiques classiques (performance, lisibilité du code...). Ces caractéristiques sont représentatives de la manière dont le logiciel a été développé. Elles influent ensuite sur le comportement du logiciel (par exemple des bugs ou alors une satisfaction de l'utilisateur). Le développeur doit donc implémenter des bonnes pratiques, suivre des guidelines ou implémenter des design patterns qui permettent d'agir sur ces caractéristiques. Dans le cadre de l'éco-conception, ces bonnes pratiques sont les green design patterns (Voir Chapitre « Développement » p.163). La norme ISO 25000 définit 8 caractéristiques. La complexité est de les optimiser. En effet, il est possible que certaines caractéristiques soit non complémentaires. Exemple : faire tourner le plus rapidement une tâche ne va pas forcément diminuer la consommation. En effet il va falloir utiliser plus de ressources pour aller plus vite.

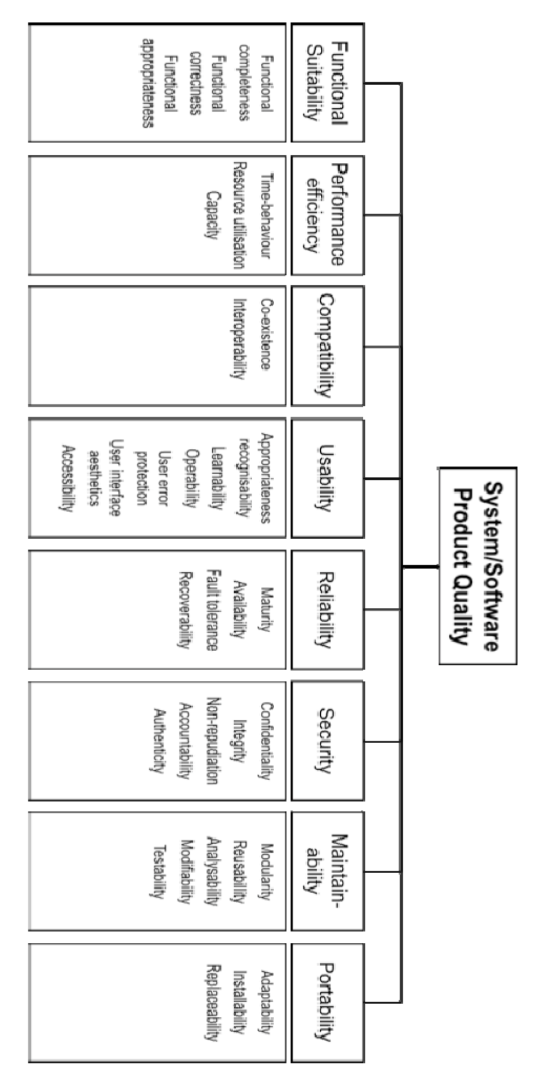


Figure 14 - Sotware product

Efficience de la performance

L'efficience de la performance est un aspect des plus importants de l'éco-conception car il va permettre de réduire la consommation d'énergie et de diminuer le besoin en matériel. En suivant les sous-catégories ISO 25022, on a donc :

- ✓ Comportement temporel : Le logiciel doit réaliser des tâches le plus rapidement possible.

- ✓ Utilisation des ressources : Le logiciel doit utiliser le moins de ressources directes (comme l'espace mémoire) possible mais aussi de ressources indirectes comme l'énergie ou les ressources humaines.

- ✓ Capacité : La capacité maximale (Vitesse CPU requise…) doit être assez importante pour que le logiciel puisse évoluer sur les futures plateformes mais à l'opposé, il ne doit pas demander une capacité importante pour fonctionner.

Voir chapitre « Green Patterns » p 163.

Compatibilité / Coexistence

Le logiciel doit fonctionner et respecter les exigences initiales sans perturber les logiciels qui se trouvent dans son environnement. Deux types de coexistence sont à prendre en compte :

- ✓ Coexistence avec la gestion d'énergie de l'OS (que l'on peut appeler efficacité de prise en compte du contexte).

- ✓ Coexistence avec les autres logiciels

L'efficacité de prise en compte du contexte est importante car le logiciel doit savoir identifier l'état du système pour agir

correctement.

Voir chapitre « Efficacité d’architecture » p 163.

Utilisabilité

La facilité d’utilisation du système va permettre à l’utilisation d’être efficace et donc de réaliser ses tâches en utilisant le moins de ressources nécessaires. L’ergonomie et l’accessibilité comme définit par le W3 entre aussi en compte dans cette catégorie. Voir chapitre « Intégration du pilier social» p.82.

Fiabilité

La robustesse aux erreurs et la confiance que l’on a dans le logiciel évite que le logiciel ne pollue le reste du système et rende le système non performant. Dans cette catégorie, on peut aussi intégrer la disponibilité du système.

Cette catégorie peut permettre un ajustement de la qualité de service (voir paragraphe phase utilisation). En effet, il n’est pas nécessaire de fournir par exemple une disponibilité haute à l’utilisateur dans tous les cas. Augmenter la sureté et la disponibilité de l’application est coûteux en ressource car cela nécessite des mécanismes lourds (redondance…).

Voir chapitre « Qualité logicielle » p. 208.

Fonctionnalité

La caractéristique de fonctionnalité permet de s’assurer que le logiciel fonctionne correctement en fonction des exigences. Du point de vue éco-conception, cela permet d’éviter des bugs trop nombreux et donc une inefficacité. Un rejet du logiciel sera alors possible par l’utilisateur. Voir chapitre « Qualité logicielle » p. 208

Maintenabilité

Cette catégorie est divisée en des sous-catégories suivantes :

- ✓ Modularité
- ✓ Réutilisabilité
- ✓ Analysabilité
- ✓ Modificabilité
- ✓ Testabilité
- ✓ Portabilité
- ✓ Adaptabilité
- ✓ Remplaçabilité

Ces catégories ont un sens encore plus important dans l'éco-conception que dans l'ISO 25000. En effet ces catégories permettent de rendre un logiciel soit maintenable et évolutif. Ceci va donc dans le sens d'un logiciel durable. Cependant face au besoin d'adapter le logiciel aux besoins réels de l'utilisateur, il est nécessaire de rendre le logiciel modulaire en installation. En effet, il faut permettre à l'utilisateur de n'installer que des fonctionnalités qu'il souhaite pour réduire le besoin en ressources. Voir chapitre « Maintenance » p. 232.

Portabilité

C'est la capacité d'un logiciel à fonctionner sur des plateformes matérielles et des environnements logiciels différents. La portabilité permet d'augmenter la durabilité du logiciel. Toutes les pratiques de ce livre allant dans ce sens d'une amélioration de la portabilité augmentent cette capacité. Voir par exemple le

chapitre « Scalabilité » p. 131 pour la portabilité des applications sur des plateformes « light ».

Securité

Il s’agit de la capacité à être protégé des tentatives extérieures d’intrusion. Cette capacité est peu liée au développement durable.

Produit système

Au sens ISO 25023, les mêmes caractéristiques que pour le produit logiciel sont applicables. C'est-à-dire que l’on doit appliquer les mêmes évaluations des capacités au niveau du système qu’au niveau du logiciel.

Ceci prend encore plus de sens quand on prend en compte un système comme le cloud computing. Il est nécessaire d’optimiser le logiciel qui tournera sur le PC client mais le même travail doit être fait sur le serveur.

Phase d’utilisation

Définition au sens ISO 25000

La qualité d'usage est le degré auquel un produit ou un système peut être utilisé par des utilisateurs spécifiques pour répondre à leurs besoins pour atteindre les objectifs spécifiques avec efficacité, efficience, absence de risque et de satisfaction dans des contextes spécifiques d'utilisation. On peut rattacher cette notion à la notion de qualité de service que l'on retrouve dans le domaine des réseaux.

Ces caractéristiques sont visibles directement par le client. Au sens environnemental, on peut retrouver ici les impacts directs : consommations en énergie, obésiciel...

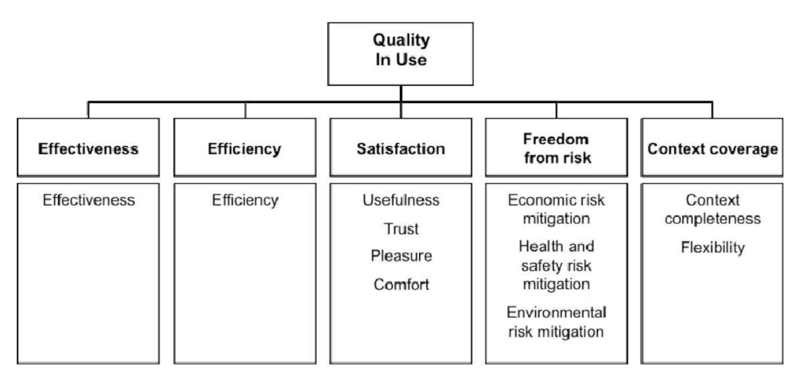


Figure 15 - Qualité d'usage

Efficacité et Efficience

L'efficacité énergétique d'une application est principalement améliorée avec des green patterns. Voir chapitre « Green Patterns » p. 163.

Satisfaction

La satisfaction de l'utilisateur n'est pas un critère qui est réellement pris en compte dans tous les développements. Il est cependant dimensionnant pour le logiciel… Voir chapitre « Identifier le besoin » p 108.

Absence de risque

Les impacts du logiciel listés dans ce manuel sont en grande partie des impacts indirects sur l'environnement : obsolescence du matériel, consommation d'énergie… Cependant, le logiciel peut être directement la cause de pollution. En effet, le logiciel agit sur l'environnement et peut donc créer des pollutions. La norme ISO 25000 évalue cet impact avec la caractéristique « Absence de risque » avec les risques environnementaux, économiques et de santé. Les risques vont dépendre du logiciel et de son objectif. Pour un logiciel de gestion financier, le risque sera plutôt économique. Pour un logiciel embarqué dans du

matériel médical, le risque sera sur la santé de l'utilisateur. Ces risques sont généralement évalués et pris en compte de façon réglementaire. On peut citer par exemple pour les logiciels médicaux la norme CEI 60601 et CEI 62304. Nous n'entrerons pas dans ce manuel dans les spécificités et la mise en œuvre de ces normes. Il est nécessaire cependant de se pencher sur certains risques. En effet, les moyens de communication utilisés par l'informatique sont de plus en plus non filaires et emploie des technologies radios. Or ces dernières sont souvent très controversées. Beaucoup d'études sont publiées mais il n'existe pas encore de consensus. Le principe de précaution fait donc loi dans la conception des logiciels gérant ces modes de communication.

Couverture du contexte

La norme ISO 25010 définit les sous-catégories suivantes :

- ✓ Complétude du contexte : degré selon lequel un produit ou un système peut être utilisé avec efficacité, efficience, absence de risque et de satisfaction dans tous les contextes d'utilisation spécifiées

- ✓ Flexibilité : dans quelle mesure un produit ou un système peut être utilisé avec efficacité, efficience, absence de risque et de satisfaction dans des contextes autres que celles spécifiées initialement dans le cahier des charges

En termes d'éco-conception il est intéressant de prendre en compte ces catégories : un logiciel doit pouvoir s'adapter à n'importe quel contexte d'utilisation en respectant les mêmes caractéristiques de consommation d'énergie. Par exemple : un logiciel devra pouvoir s'adapter sur n'importe quel OS en ayant les mêmes performances ressenties par l'utilisateur. Voir chapitre « Scalabilité » p.131

Utilisateurs

Différents niveaux d'utilisateurs sont identifiés dans la norme ISO 25000:

- ✓ Utilisateurs principaux : Personnes qui interagissent avec le système pour effectuer les fonctionnalités principales

- ✓ Utilisateurs secondaires : Utilisateurs qui fournissent du support :

    - o Fournisseurs de contenu, managers système, administrateurs,

    - o Maintenance, intégrateurs

- ✓ Utilisateurs indirectes : personnes qui reçoivent les sorties du logiciel mais qui n'interagissent avec le système.

Il en va de même pour l'éco-conception des logiciels. On peut ensuite appliquer chaque caractéristique aux différents utilisateurs (ainsi qu'à leurs besoins) : Par exemple pour l'efficacité énergétique d'un logiciel bureautique:

- ✓ Maîtrise de la consommation pour que l'utilisateur rédige un document

- ✓ Maîtrise de la consommation pour que l'ingénieur système déploie et maintienne le logiciel dans le service informatique

- ✓ Maitrise de la consommation pour les équipes qui maintiennent le développement du logiciel

Pour aller plus loin

ISO/IEC 25000:2005 - Ingénierie du logiciel -- Exigences de qualité du produit logiciel et évaluation (SQuaRE) -- Guide de SQuaRE

Pilotage énergétique des projets informatiques

Le pilotage des projets informatique se fait habituellement en fonction de nombreux paramètres très répandus (planning, charge de développement…). Un des indicateurs les plus utilisé pour connaître le coût de développement est le taux journalier moyen. De ce TJM, on déduit ensuite les coûts de projet ou alors on essaye d'optimiser ce coût. Par exemple avec un TJM de 300 €, un projet sur lequel 2 personnes ont travaillé 50 jours coûtera 30 000 €. Quel serait alors un équivalent en gestion du point de vue éco-conception ?

Les indicateurs en green IT ne sont pas actuellement encore matures, et de plus, ne permettent pas de couvrir tous les cas d'application, et notamment l'éco-conception, en particulier en ce qui concerne la consommation énergétique. Au niveau des postes de travail, la consommation ainsi que le taux de veille peuvent être utilisés et parfois standardisés (comme le TEC d'Energy Star). Ensuite, on passe directement au PUE qui convient en partie aux data centers mais pas forcément aux autres infrastructures comme les bureaux.

L'infrastructure d'un bureau est le bon exemple du manque d'indicateur. Malgré le nombre important des bureaux, les seuls indicateurs que nous trouverons seront soit la facture d'électricité soit la consommation du parc. L'équivalent émission CO^2 lui ne permet pas de gérer précisément l'efficacité et la consommation énergétique. Alors que faire ? Voici une réflexion sur un indicateur potentiel.

Pour étude, nous pouvons prendre un bureau avec les caractéristiques suivantes :

- ✓ 8 personnes,
- ✓ 10 PC standards, consommant en moyenne 100 Wh et activés en journée,
- ✓ Une imprimante consommant en moyenne 200Wh allumée tout le temps,
- ✓ Un serveur allumé 24h/24h consommant en moyenne 500Wh.

La consommation hebdomadaire est donc de 167 kWh. En associant le nombre de personnes, on déduit que l'on a une consommation de 4,2 kWh par personne et par jour. On a ainsi intégré, pas uniquement la consommation du PC, mais aussi la consommation de l'infrastructure. De plus le coût de la consommation la nuit et le week-end a été pris en compte.

La complexité sera ensuite de pondérer les ressources partagées (par exemple un serveur partagé entre différents services). Dans un premier temps une simple division de la puissance consommée sera suffisante. Pour la consommation externe induite (cloud, web…), le calcul est plus compliqué. De nombreux essais de calcul existent … en particulier sur greenit.fr (pour Google par exemple). Compte-tenu de la complexité, il ne faut prendre en compte dans un premier temps que les applications métiers.

Le calcul du coût énergétique d'un projet va pouvoir être fait à partir de ce TJEM (appelons le « taux journalier énergétique moyen»). Si l'on a le TJEM des différents services qui ont travaillé sur le projet alors il est facile de calculer le coût global du projet en fonction des ressources utilisées et du nombre de jours réalisés.

Cet indicateur ne remplacera pas les autres indicateurs plus classiques mais permettra d'avoir une notion sur l'énergie qui est utilisée pour développer un logiciel.

Idées d'action Green Code Lab :

Evaluer l'intégration de l'éco-conception dans d'autres processus et normes : ITIL, CMMI, processus métier…

EXPRESSION DE BESOIN

Identifier le besoin

Optimisation de la qualité de service

Un logiciel est destiné à un utilisateur. Il a pour but de répondre à différents besoins. L'adéquation entre les besoins exprimés par l'utilisation, ce qui est réellement implémenté dans le logiciel et ce que veut réellement l'utilisation est une problématique complexe. Selon une étude de l'institut Standish Group, 45% des fonctionnalités demandées par des utilisateurs ne sont jamais utilisées. L'implémentation d'une multitude de fonctionnalité dans les logiciels est malheureusement la solution choisie par les éditeurs de logiciel pour éviter de passer à côté de besoins ou pour valoriser les nouvelles versions et leur acceptation par leurs clients. C'est une des causes de l'obésiciel compte tenu de la multitude de fonctions et d'interface que le logiciel intègre. On peut aussi regarder ce problème en analysant le niveau de service qu'offre l'application: la qualité de service. Elle détermine un degré de service fourni à l'utilisateur. Dans un autre domaine qu'est le réseau on parle par exemple de qualité de service des données : disponibilité, perte, robustesse… La qualité de service peut aussi être optimisée dans le domaine logiciel.

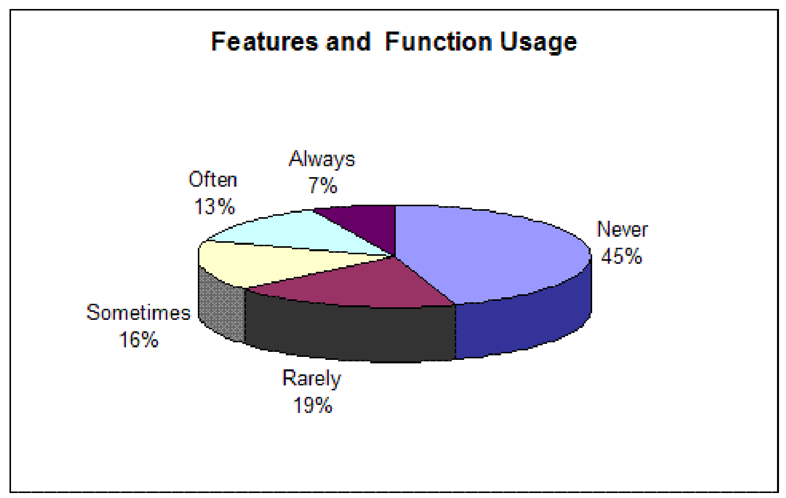


Figure 16 - Etude standish group

Comment adapter cette qualité de service ? En fournissant des fonctionnalités plus ciblées. Microsoft a par exemple évalué qu'en fournissant 20 résultats à l'utilisateur du moteur de recherche Bing, la consommation était diminuée de 80%[28] avec un service équivalent. Cette conclusion peut être appliquée à de nombreux logiciels : suite bureautique, logiciel pour mobile… Les bénéfices sont alors nombreux :

✓ Moins d'espace mémoire nécessaire,

✓ Performance accrue et donc consommation diminuée,

✓ Satisfaction de l'utilisateur(Meilleure lisibilités des résultats…).

Les risques existent cependant car on ne va peut-être pas répondre aux besoins de tous les utilisateurs. La solution est alors de fournir la possibilité à l'utilisateur d'augmenter couverture fonctionnelle du logiciel :

[28] http://research.microsoft.com/pubs/101217/Green_paper.pdf

- ✓ Installation de modules supplémentaires pour augmenter les capacités du logiciel (voir chapitre Conception),

- ✓ Option pour afficher des fonctionnalités avancée.

Expression de besoin environnementale

En plus de la nécessité de répondre correctement aux besoins fonctionnels des utilisateurs, l’aspect environnemental à proprement parler commence à entrer dans les demandes des entreprises et collectivités. Que ce soit dans les réclamations des utilisateurs « classiques » ou dans leurs critères de choix ou alors dans les appels d’offres des sociétés ou organismes publiques, les besoins sont assez flous voire non directement exprimés. Cela est normale car l’éco-conception et le développement durable sont des habitudes qui ne sont pas entrées dans la « conscience collective », encore moins pour le logiciel où l’éco-conception est un domaine tous nouveau.

C’est du devoir d’éthique du développeur de reformuler ce besoin en expression claire et vérifiable par l’utilisateur. Cette éthique est malmenée ces dernières années par certaines sociétés avec le buzz de la green IT. Surfant sur le flou, il est facile de mentir, de cacher la vérité ou de faire des allégations sur des bénéfices environnementaux non prouvés. C’est ce que l’on appelle le Greenwashing.

Donc attention lors de la communication autour de ce besoin et des caractéristiques d’éco-conception de ne pas tomber dans le piège du greenwashing. Pour vous aider, voici les sept péchés capitaux énoncés par la société TerraChoice (Source Greenit.fr)

Dans plusieurs rapport et sur son site SinsofGreenWashing.org, la société TerraChoice, qui gère l’écolabel canadien de type I « Ecologo », a défini 7 péchés de greenwashing. Nous les avons repris et avons tenté pour chacun d’entre eux de les illustrer par des exemples tirés du secteur des Technologies de l’Information.

1 - Péché du compromis caché

Toute prétention indiquant qu'un produit est « vert » mais n'étant fondée que sur un nombre déraisonnablement restreint d'attributs en occultant d'autres enjeux environnementaux importants. Exemple : les publicités d'appareils électroniques dits « écologiques » car économes en énergie occultent le plus souvent l'impact environnemental de la fabrication (énergie grise et pollutions chimiques) et de la fin de vie (le produit est peut -être plus compliqué à recycler et contient des matières dangereuses). Dans l'étude menée par Terrachoice en 2009, 73% des produits évalués sont coupables de ce péché.

2 - Péché d'absence de preuve

Toute prétention environnementale qui ne peut être étayée par une information facilement accessible, ou par l'agrément d'une tierce partie. Exemple : les équipements électroniques ou informatiques qui avancent « une économie d'énergie de 5% » sans preuve de leur prétention ou agrément.

3 - Péché d'imprécision

Toute prétention mal définie ou dont la définition est si vague qu'elle peut prêter à mauvaise interprétation par le consommateur ciblé. Exemple : l'expression « sans substances nocives » ne veut rien dire, selon la quantité, toute substance peut devenir nocive. les expressions « vert », « sans danger pour l'environnement » ou « préserve l'environnement » ne veulent rien dire sans explications détaillées.

4 - Péché de non pertinence

Toute prétention environnementale qui, bien que vraie, est inutile ou insignifiante pour le consommateur eco-responsable, le détournant ainsi d'un meilleur choix. Exemple : Les produits qui mettent en avant leur conformité ROHS, cette précision est inutile puisque pour commercialiser un produit en Europe, le

respect de cette directive est obligatoire depuis 2003.

5 - Péché du moindre des deux maux

Toute prétention environnementale qui peut se vérifier dans une catégorie de produits, mais qui pourrait détourner l'attention du consommateur sur les impacts environnementaux de l'ensemble de la catégorie. L'exemple que donne Terrachoice est celui de la cigarette à base de tabac provenant de l'agriculture biologique qui pourrait sous-entendre que le produit est sain alors que globalement la cigarette est nocive pour la santé. Dans le secteur des technologies de l'information, on pourrait citer l'exemple de Microsoft qui, pour la promotion de Microsoft Windows 7, met en avant ses meilleures capacités de gestion d'énergie, alors que globalement, ce système d'exploitation nécessite un ordinateur 243% plus performant que pour faire tourner Microsoft Windows XP ou certaines distributions GNU/Linux qui suffisent pour un usage bureautique.

6 - Péché du faux ecolabel

Lorsqu'un produit, par le biais de mots ou d'un logo, veut faire croire qu'il est agréé par un éco-label. Les exemples sont nombreux dans le secteur des technologies de l'information : Fujitsu Siemens avec son label « Green IT » ou encore NEC avec son label EcoGreenIT ou même ECOSustainability.

7 - Péché du mensonge

Toute prétention environnementale qui, après vérification, s'avère fausse. Exemple : publicité d'un produit informatique qui affiche le logo Energy Star ou EPEAT sans être certifié. Dans l'étude de TerraChoice, moins de 1% des produits évalués sont concernés par ce péché.

Pour aller plus loin

Site Terrachoise - http://sinsofgreenwashing.org/

Rubrique Greenwashing de GreenIT.fr : http://www.greenit.fr/tag/greenwashing

Trophées Pinochio du développement durable – Les amis de la terre - http://www.prix-pinocchio.org/

Expression de besoin en tant qu'utilisateur averti

Exprimer des besoins en termes d'éco-conception vers les développeurs est une nécessité pour faire avancer la green IT. Un développeur est avant tout un utilisateur averti, il a la connaissance « de l'intérieur du moteur », des performances de logiciel… Tel un mécanicien qui irait acheter une voiture, il est le plus à même d'exprimer un besoin en termes d'éco-conception. En effet, l'éthique des éditeurs et des développeurs ne fait pas tout et il est nécessaire pour chaque utilisateur de remonter de tels critères aux fournisseurs de logiciel. Mais pour un utilisateur ne connaissant pas le monde de l'informatique, il est facile de tomber dans les pièges de l'obsolescence ressentie et de mal identifier un logiciel non respectueux de l'environnement (voir Chapitre « Le bloatware » p. 44). C'est donc un devoir des développeurs d'exprimer ces requêtes par tous les moyens possibles : forum utilisateurs, système de remonté de bug… Les logiciels libres permettent d'autant plus facilement ces pratiques que le logiciel est ouvert et peut être modifié. Pour des modifications profonde de ces logiciels, un travail avec la communauté sera nécessaire. Pour des évolutions mineures mais bénéfiques, la contribution d'un module ou d'une partie éco-conçu sera possible. Soyez libre de rendre le logiciel libre plus respectueux de l'environnement.

Choisir le bon modèle économique

Les 4 modèles économiques de développement logiciel

Quels sont les modèles économiques possibles pour un logiciel ? Dans le livre «L'économie des logiciels » , François Horn distingue 4 modèles économiques[29] :

	Absence de standardisation	Standardisation du produit et/ou des composants
Produits Dédiés	Monde interpersonnel : Logiciels sur-mesure développés « exnihilo »	Monde de la production flexible : Logiciels standards et services sur mesure. Logiciels sur mesure à partir de composants standardisés.
Produits Génériques	Monde de la création : Logiciels libres (code source)	Monde fordiste : Progiciels commerciaux (code objet et services limités)

Figure 17 - Les 4 modèles économiques

Monde Interpersonnel

Horn définit ce monde comme celui de la production d'applications spécifiques par des services informatiques. Ces logiciels ont une fiabilité haute. L'adéquation aux besoins des

---

[29] http://matisse.univ-paris1.fr/fr/IMG/pdf/eco_logiciels_reperes_chapitre_5-1.pdf

utilisateurs est selon lui variable en fonction du niveau d'intercompréhension entre l'éditeur et l'utilisateur.

Monde de la production flexible

Un exemple de logiciel issu de ce monde est un système de gestion d'entreprise. Horn considère que les principaux producteurs sont les S.S.I.I., les sociétés de conseil, et sont liées à des producteurs de matériel ou de progiciel. Selon lui la productivité et la fiabilité des logiciels sont assez élevées. L'adéquation aux besoins des utilisateurs dépend de la qualité de la relation de service comme pour le monde interpersonnel.

Monde de la création

Ce monde rassemble le monde que l'on appelle « open » ou « libre » (Par exemple le système d'exploitation GNU/Linux). Les principaux producteurs sont les universités, les centres de recherche, les créateurs indépendants. La productivité selon Horn est variable (en fonction de la popularité, de l'importance de la communauté…). Cependant la fiabilité des logiciels est faible mais une amélioration rapide est possible (pour les logiciels qui connaissent un succès initial). L'adéquation aux besoins des utilisateurs est forte pour la communauté informatique mais plus problématique pour les simples usagers. Se pose pour ce monde le problème de la gratuité. Beaucoup discutée il est difficile de dire si ce modèle est durable économiquement.

Monde Fordiste

Ce monde englobe la production de logiciel (comme par exemple les logiciels bureautique) par les « Editeurs » de logiciels. Les utilisateurs principaux sont les ménages, les entreprises et les administrations. La productivité est élevée mais la fiabilité est souvent insuffisante. L'adéquation aux besoins des utilisateurs existe uniquement pour des besoins standards.

Quels modèles économiques pour un logiciel durable ?

Pour pouvoir répondre à cette question, il faut d'abord répondre aux questions du début du chapitre.

Le modèle répond-t-il bien aux besoins des utilisateurs et offre-t-il des choix alternatifs ? : Les mondes de la production flexible et interpersonnelle permettent une adéquation assez haute aux besoins des utilisateurs. Cependant ceci est à contrebalancer avec le fait que ces mondes travaillent pour un nombre restreint d'utilisateur. Il n'adresse que très peu les particuliers. Le monde du fordisme lui au contraire tente de répondre à tous, d'où son défaut de répondre à des besoins trop standard. Le monde de la création, lui, souffre du mal du passionné d'informatique : les logiciels sont développés par des informaticiens pour des informaticiens et ne répondent pas assez aux besoins de tous. Une réponse universelle est difficile mais le défi est, quelque soit le monde économique où l'on est, de tendre vers cette réponse et d'être conscient des lacunes.

Le modèle permet-il la production d'un logiciel respectant l'environnement ? Cette question est difficile. Une réponse est de dire que si un processus est productif il pourra absorber la prise en compte des exigences environnementales. A cela la capacité à produire un logiciel fiable s'ajoute. Dans les différents mondes la fiabilité et la productivité fluctuent en fonction de nombreux paramètres. Il en ressort cependant que certaines clés sont nécessaires. La réutilisabilité du logiciel est importante pour augmenter la fiabilité et la productivité. La standardisation des pratiques et des langages est un aspect important pour éviter des gaspillages. L'amélioration de la fiabilité et de la productivité est une question économique « il faut mettre les moyens ». Cependant les gains ne sont pas immédiats et c'est la raison pour laquelle peu d'améliorations sont mises en place. Pour le monde de la création, le problème est différent. Il n'est pas organisé comme une société mais comme une communauté. La productivité et la fiabilité vont

dépendre de l'importance de la communauté, des règles mises en place par les leaders des communautés.

Le modèle économique est-il viable pour l'entreprise ? A cette question, les mondes interpersonnels, flexible et fordiste semblent répondre positivement. Cependant, si l'on revient à la question précédente, on peut moduler cette réponse : L'intégration de critères environnementaux dans ces mondes diminuerait peut-être les marges des sociétés. Cela est vrai à première vue mais les bénéfices à moyen terme vont permettre de trouver un gain. Nous avons en effet vu dans le chapitre « Quick And Dirty » p.86. Mais attention, tous les éditeurs ne sont pas égaux. Certains gros éditeurs affichent des marges très importantes. Microsoft par exemple pour Microsoft Windows et la suite Office atteint une marge brute de 85%. Qu'en est-il de la viabilité du monde de la création ? Vaste question à laquelle il est difficile de répondre ! Les sociétés qui mettent à disposition des logiciels libres peuvent avoir plusieurs modèles économiques : ventes de service associée à la mise en place de logiciel, financement par des fond de recherche et développement ou tout simplement bénévolat (comme certaines communautés). Pour la production par des voies non commerciales, le modèle de gratuité est discuté. L'idéal serait un financement permettant de rendre plus durables les projets et permettant aussi d'intégrer plus rapidement les contraintes de fiabilité et environnementales.

Que faire concrètement ?

A moins de créer une société ou de développer son propre logiciel, le modèle est souvent imposé. Nous avons fait un tour rapide des questions à se poser et nous avons vu qu'il n'y avait pas réellement de réponse universelle. Le principe important est de répondre le plus possible aux besoins de l'utilisateur tout en intégrant les contraintes de fiabilité et environnementale. Cela demande une flexibilité importante : comment développer un système d'exploitation pour un déploiement mondial tout en répondant aux besoins spécifiques de chaque utilisateur ?

Comment Intégrer des pratiques vertes et de la fiabilité alors que j'appartiens à une communauté (et qu'il n'y pas de service qualité ou DD et que le projet n'est pas financé pour cela) ? Le défi est de faire le grand écart entre les différents mondes pour appliquer les bonnes pratiques de chacun. Nous n'avons pas de réponses claires (Pour le moment) mais il est primordial d'évaluer les modèles économiques sous l'angle du développement durable.

Libre, gratuit, open source

Libre d'utiliser les logiciels

Le modèle libre est mal compris non seulement de la plupart des utilisateurs mais aussi des personnes du monde logiciel. En effet, libre, gratuit et open source sont des termes qui ont des significations bien distinctes mais qui sont souvent associés à tort dans les esprits. Tout d'abord le terme anglais pour définir le mouvement libre est free. Or en anglais free veut dire à la fois libre et gratuit. Le mouvement libre ne prône pas la gratuité. Richard Stallman, un des fondateurs du mouvement libre, définit le libre de la manière suivante : Le logiciel libre confère à son utilisateur quatre libertés :

1. la liberté d'exécuter le programme, pour tous les usages,
2. la liberté d'étudier le fonctionnement du programme et de l'adapter à ses besoins,
3. la liberté de redistribuer des copies du programme (ce qui implique la possibilité aussi bien de donner que de vendre des copies),
4. la liberté d'améliorer le programme et de distribuer ces améliorations au public, pour en faire profiter toute la communauté.

On peut ainsi définir le mouvement libre comme une

philosophie qui faciliterait le déploiement et le partage des logiciels. L'inverse des logiciels libres sont les logiciels « propriétaires » ou logiciel « privateurs » comme le définit Richard Stallman. Pour bien comprendre la différence, Richard Stallman utilise l'image de la recette de cuisine : selon le principe du libre, si vous avez obtenu une recette par un moyen quelconque, vous avez le droit de redistribuer, modifier, vendre la recette comme vous le souhaitez alors que selon le principe privateur, vous n'avez pas accès à la recette, vous avez uniquement le gâteau fini et vous ne devez le manger que tout seul. Quand bien même vous auriez la recette, vous ne pourriez pas la modifier ou la redistribuer.

Les logiciels non libres

Beaucoup de type de logiciel sont associés à la mouvance à tort :

- ✓ Freeware (ou Gratuitiel) : Le libre n'interdit pas la revente. Les freewares ne sont pas libre car il s'agit pour l'auteur de diffuser son logiciel mais il n'ouvre pas l'accès au code du logiciel

- ✓ Shareware (partagiciel) : comme le freeware, l'utilisateur ne peut modifier ou avoir accès aux sources. L'accès est généralement payant si on le souhaite ou il est limité si on ne paye pas.

- ✓ Open source : Un logiciel open source est un logiciel dont on a accès aux sources. La liberté de distribuer ou de modifier le code n'est pas forcément autorisée. Il y a un désaccord entre les mouvements Libre et Open source mais des travaux d'harmonisation sont en cours

Pour aller plus loin

Livre Richard Stallman et la révolution du logiciel libre – Eyrolles

Free Softwaare Foundation – www.fsf.org

Installation modulaire

Nous avons vu dans ce chapitre qu'une des solutions pour réduire l'impact du logiciel était de réduire ses fonctionnalités. Ceci peut être réalisé via une installation modulaire : les fonctionnalités du logiciel uniquement nécessaires à l'utilisateur sont alors installées, les autres fonctionnalités non nécessaires ne sont pas mises sur le disque dur. On diminue ainsi l'espace occupé et aussi les ressources nécessaires (CPU, mémoire...). C'est finalement l'esprit initial des distributions GNU/Linux.

Ce fonctionnement existe déjà dans certains logiciels :

- ✓ Linux avec les paquets et les modules du noyau
- ✓ Extensions pour les navigateurs internet (Chrome, Firefox...)
- ✓ Modules pour le serveur web Apache
- ✓ Librairies permettant d'ajouter des fonctionnalités de haut niveau à un langage de programmation

Cependant des améliorations sont à fournir sur certains logiciels et doivent être généralisés car il sont assez limités et ne permettent pas une optimisation substantielle car :

- ✓ Les modules pouvant être désinstallés sont souvent des

modules spécifiques et ne touchent pas au cœur du logiciel. La désinstallation ne permet alors pas une optimisation importante.

- ✓ Le choix des modules n'est pas simple et nécessite souvent une compréhension du logiciel.

- ✓ L'accès à l'installation et à la désinstallation n'est pas simple, les utilisateurs préfèrent donc installer le maximum de modules.

Il est donc nécessaire d'offrir des interfaces plus simples pour l'utilisateur. Ceci rendra l'acte d'installation et de désinstallation plus simple. Le regroupement de modules permettra de plus d'offrir des modes de désinstallation comme des modes d'économie d'énergie sont proposés selon des profils d'utilisation. Par exemple : Basique, Jeux, Internet, novice...

Ce fonctionnement demande cependant de grands changements dans l'architecture du logiciel. En effet, la conception des logiciels intègre une architecture avec des modules délimités mais pas forcément indépendants. Un module doit donc être le plus indépendant possible des autres modules. La fonctionnalité qui sera absente n'impactera donc pas les autres modules. Si il y a des dépendances, la désinstallation des autres modules devra donc être faite avec celle du module (D'où la nécessité d'offrir des modes de désinstallation).



Développement

Cœur du métier logiciel, des bonnes pratiques de développement sont les clés pour obtenir un produit respectant les critères du développement durable. Les green design patterns, motifs d'éco-conception des logiciels, répondent en partie à cela.

INTRODUCTION

Principe des green design patterns

Un design pattern est un motif de conception logiciel réutilisable permettant d'atteindre un certain but. Il n'est pas codé mais peut être implémenté rapidement dans un code. Généralement ces patterns sont adaptés à la programmation orientée objet et décrivent les interactions possibles entre classes et objets. On retrouve cependant des design pattern dans d'autres domaines. Pour ce qui est de l'architecture du code, on désignera plutôt les solutions comme des « Architecture Pattern ». On regroupe généralement les motifs de conception logiciels dans les catégories suivantes :

✓ Creational patterns : Mécanisme de création des objets

✓ Structural patterns : Interactions entre les éléments du code

✓ Behaviour patterns : comportement des éléments

L'objectif de l'éco-conception des logiciels n'est pas de calquer totalement ces patterns mais de s'inspirer de ces principes. En effet, la critique qui est faite sur ces patterns est qu'ils ne sont pas assez génériques. Cependant il est intéressant d'offrir aux développeurs des principes généraux qui pourront être appliqués simplement lors de la programmation.

On peut alors définir un green design pattern de la manière suivante :

Un green design pattern est un motif de conception logiciel réutilisable qui réduit l'impact environnemental du système sur lequel il sera installé.

Il faut cependant faire attention au terme de green design pattern car il se restreint à l'aspect environnemental et met de côté les deux autres piliers (social et économique). On peut se poser la question si il peut exister des Economic patterns mais pour les social patterns, il est clair que cela s'applique (Voir les règles d'accessibilité). Si les green patterns ont pour vocation de répondre aux objectifs environnementaux, il est alors nécessaire d'en élargir le champ et de s'interroger systématiquement sur leur impact économique et social. Il serait en effet dommage de mettre en œuvre par exemple une solution réduisant de moitié la consommation énergétique mais qui doublent le coût du logiciel ou demande un mode d'organisation non conforme avec les exigences sociales du DD.

La complexité des green patterns est de fournir des designs qui seront génériques. La multiplicité des technologies amène à penser qu'ils y aura des green patterns spécifiques et que certains patterns génériques seront dépendant d'un contexte et donc difficilement applicables.

Pour aller plus loin

Livre « Design Patterns - Elements of ReusableObject-Oriented Software » -The "Gang of Four": Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Livre "Code Complete" - Steven C. McConnell

Leviers d'amélioration

L'amélioration de l'impact environnemental nécessite de se poser deux question : quelles caractéristiques du logiciel veut-on améliorer ? Et comment peut-on les améliorer ? Les réponses à la première question sont principalement issues de l'analyse du cycle de vie : il faut diminuer les pollutions en phase de conception, réduire la consommation en phase

d'utilisation et prolonger la vie du logiciel. Il n'existe pas de méthode ou de « framework » couvrant ses domaines, cependant nous allons voir une liste exhaustive d'axes permettant de diminuer l'impact du logiciel sur l'environnement. Nous verrons tout au long de cet ouvrage que ces axes sont étroitement liés mais que bien discerner chaque objectif de l'éco-conception est important pour couvrir tous les domaines.

✓ Efficience de la consommation d'énergie

- La part de consommation générée par le code doit être maîtrisée.
- Exemple : Demander que le système soit toujours dans l'état de performance le moins élevé

✓ Diminution des temps d'exécution

- Le code s'exécute plus rapidement donc libère les ressources plus rapidement
- Exemple : Si mon programme s'exécute en deux fois moins de temps, le système peut passer en veille deux fois plus de temps.

✓ Facilitation d'utilisation

- L'application améliore l'ergonomie du système donc rend plus efficace son utilisation
- Exemple : une IHM fluide et compréhensive permet à l'utilisateur de ne pas rester sur le PC à rechercher le fonctionnement. Elle limite aussi le nombre d'interactions avec l'utilisateur.

✓ Diminution des ressources nécessaires

  - o Demander moins d'espace mémoire et CPU pour éviter l'obésiciel

  - o Exemple : Inflation de l'espace nécessaire pour Microsoft Windows et entropie

- ✓ Maîtrise de l'impact lors de la « fabrication » du programme

  - o Un outil responsable conçu avec un outil tout autant responsable

  - o Exemple : développer une application qui consomme peu d'énergie à l'aide d'un PC polluant.

- ✓ Prolongation de la durée de vie du logiciel

  - o La conception du logiciel doit intégrer que le logiciel puisse durer dans le temps (jusqu'à plus de 6 ans)

  - o Exemple : Prévoir que le code soit maintenable et ne dépasse pas certaines limites en termes de ressources (CPU, RAM, etc.) même avec l'ajout de service packs ou s'assurer que les technologies sous-jacentes et choix d'architecture permettent de prolonger la durabilité du logiciel en évitant notamment les impasses technologiques ou en prévoyant l'impact des sauts technologiques à venir.

Comment atteindre ces buts ? Il est nécessaire de travailler sur plusieurs domaines que l'on nommera “domaines d'efficacité”.

- ✓ Efficacité d'architecture

  - o La façon dont est conçu le programme peut avoir une influence

  - o Dans l'architecture : Langage, répartition des calculs entre client/serveur

  - Exemple : Facebook avec le projet Hip-Hop qui compile le code PHP en C++[30]

- ✓ Efficacité de calcul
  - Un algorithme plus efficace peut permettre de diminuer l'impact
  - Exemple : bufferisation d'une lecture DVD
- ✓ Efficacité de donnée
  - La gestion des données peut impacter l'efficacité et la place du code
  - Exemple : Optimisation de la localité des données comme l'utilisation du cache plutôt que des données en RAM.
- ✓ Efficacité de prise en compte du contexte
  - Un logiciel n'est pas une unité indépendante
  - Il doit communiquer avec l'extérieur
  - Un programme « context-aware » sera plus intelligent et prendra mieux en compte les impacts environnementaux
- ✓ Efficacité des outils
  - Les logiciels et outils permettent de rendre le processus de fabrication efficace.

[30] http://www.greenit.fr/article/energie/hiphop-for-php-facebook-veut-reecrire-php

Chaque domaine d'efficacité va permettre de couvrir les objectifs de l'éco-conception. Voici le mapping Moyen / Domaine. Les cases grisées permettent d'identifier l'impact des axes sur les objectifs de l'éco-conception. Si la case n'est pas grisée, l'impact de l'axe est considérée comme faible ou nulle pour l'objectif.

	Efficacité d'architecture	Efficacité de calcul	Efficacité de données	Prise en compte du contexte	Efficacité des outils
Réduction de la consommation d'énergie	■	■	■	■	
Augmentation de la performance	■	■	■		
Facilitation d'utilisation				■	
Diminution des ressources nécessaires	■		■	■	
Maîtrise de l'impact lors de la « fabrication »					■

Figure 18 - Mapping des patterns

Idée Green Code Lab

Ce framework est un début de réflexion. Pourquoi ne pas participer à son amélioration ? Phases de vie, impacts économiques, environnementaux et sociaux sont à ajouter par exemple.

LIEN ENTRE MATERIEL ET LOGICIEL

Scalabilité

La course en avant

Plus un logiciel est complexe, intègre de fonctionnalités ou doit gérer un nombre important d'utilisateurs et plus il nécessite de ressources matérielles. Nous avons vu ce concept dans le chapitre « Évolution du logiciel » p.40. Une des solutions retenues par le monde du logiciel est donc d'augmenter les ressources matérielles lorsqu'il est nécessaire d'implémenter de nouveaux aspects dans le logiciel.

Cela ne pose pas de problème dans la plupart des cas car les concepteurs profitent de certaines ruptures pour changer de matériel :

- ✓ Changement de technologie comme le passage de la 3G à la 4G

- ✓ Changement de version du logiciel (Microsoft Windows XP vers Seven par exemple)

Cela pose donc un problème d'obsolescence du matériel car les changements technologiques ou les améliorations de fonctionnalité s'accompagnent presque toujours d'un changement de matériel.

Cependant ce problème de limite de ressources est en partie traité quand il se produit pendant la phase d'utilisation du logiciel. En effet, sans aucune rupture ou évolution majeure, le changement de matériel ne peut être justifié. Il pourrait de plus être très couteux. Prenons le cas d'un logiciel conçu initialement avec une estimation de fonctionnalité ou d'utilisateur. Les concepteurs vont donc dimensionner le

matériel requis en fonction du logiciel qui sera nécessaire (technologie, estimation du nombre de ligne de code, mémoire nécessaire). Une marge sera ajoutée pour absorber un peu un potentiel succès et se donner du temps. Mais qu'arrive-t-il si ces hypothèses sont fausses et que par exemple le nombre d'utilisateurs est supérieurs aux attentes ? Oui le succès peut arriver ! Et bien, nos chers ingénieurs ont inventé la capacité de scalabilité des applications. Mais utiliser cette capacité est-ce vraiment une solution durable (du point de vue environnemental bien-sûr) ?

Scalabilité ou la course à l'armement

La scalabilité, c'est la qualité d'une application à fonctionner dans les mêmes conditions de fonctionnement (Performances, fonctionnalités, disponibilités…) lors d'une augmentation de la charge d'utilisation (nombre de données ou d'utilisateurs par exemple). La scalabilité est aussi nommé élasticité ou capacité de mise à l'échelle. On parle beaucoup de scalabilité pour les applications web car elles doivent absorber un nombre important et non connus à l'avance d'utilisateur.

La modification de la scalabilité se fait par définition en augmentant les ressources matériel et cela sans changement majeur du logiciel. On distingue deux types de solution :

- ✓ Verticale : Capacité à augmenter les ressources du serveur en ajoutant de la mémoire, en changeant le processeur, en rajoutant des disques…

- ✓ Horizontale : Capacité à augmenter le nombre de ressources matérielles. La répartition des charges se fait grâce à un répartiteur (load balancer).

Pour la scalabilité verticale, la relation entre ressources et capacités est « normalement linéaire » (Il peut en effet avoir des effets de seuil lié aux ressources matérielles disponibles). C'est-à-dire que si on doit doubler un nombre d'utilisateurs, il

faudra doubler les capacités des ressources matérielles. Se pose alors un problème financier car cette multiplication de ressources, elle, n'est pas linéaire. Mais surtout, l'augmentation des ressources n'est pas illimitée. Même si l'application est « scalable », on va être confronté à des barrières technologiques et des goulots d'étranglement. Une de ces limitations est la limitation des 10 000 connections simultanées, appelé C10K. Dank Kegel a posé le problème en 2006[31] à partir du constat que les systèmes d'exploitation actuels et la conception des applications client-serveur ne permettaientt pas de répondre à la connexion simultanée de 10 000 utilisateurs. En effet, l'architecture des serveurs avec en particulier la gestion des entrées-sorties et les modèles de communication concurrente limite les performances. Des solutions existent bien-sûr pour contourner ces limitations :

- ✓ Framework javascript serveur Node.js
- ✓ Serveurs web léger tels que Nginx, lighttpd…

De façon générale, gérer l'impact de la scalabilité verticale n'est pas simple. Pour éviter un certain bloatware et une multiplication continuelle des besoins matériels, il faut repenser les architectures logicielles et les concepts existants. En effet sans solutions d'architectures et techniques adaptées, le bloatware arrivera avec l'augmentation verticale des applications à cause de limitations technologiques (tels que goulots d'étranglement de la mémoire ou le problème C10K vu précédemment). La linéarité entre utilisateurs/fonctionnalités et ressources matérielles ne sera alors plus applicable. Un logiciel mal conçu pourrait consommer quatre fois plus de ressources avec deux fois plus d'utilisateurs. Les solutions utilisées pour offrir une meilleure scalabilité verticale doivent donc être durables.

Pour ce qui est de la scalabilité horizontale, l'impact est

---

[31] http://www.kegel.com/c10k.html

similaire. La barrière n'est plus logicielle mais plutôt matérielle. En effet, si les besoins en charge augmentent, il suffit « juste » de rajouter des unités. Cette solution est du point de vu environnemental (et économique) une sorte de fuite en avant car elle ne traite pas le problème de l'entropie du logiciel. Les problématiques alors induites sont celle des data centers : climatisation, énergie…

Nous le voyons donc, la scalabilité, qu'elle soit verticale ou horizontale, ne répond pas forcément à l'entropie et à l'obésité du logiciel. Certaines solutions permettent de passer des barrières technologiques mais jusqu'à quand ? Il faut donc réfléchir à d'autres solutions.

Scalabilité inverse

La scalabilité implique un concept implicite : l'application doit avoir la capacité de fonctionner sur des nouvelles plates-formes plus puissantes. Personne ne voudrait installer une application moins puissante. Du point de vue environnemental et énergétique, on peut vouloir cela. En effet de nombreux processeurs moins consommateurs arrivent sur le marché. Tout d'abord dédiés pour les appareils mobiles, ils sont de plus en plus utilisés pour d'autres applications : serveurs auto-hébergés, PC « light », plug computer… On peut citer par exemple les processeurs ARM, très répandus dans ce type de matériel.



Figure 19 - Plugcomputer Seagate Dockstar

Faire fonctionner les applications existantes sur ces processeurs n'est cependant pas toujours si simple. En effet, les applications (OS, driver, framework…) sont généralement dimensionnées pour des plates-formes ayant des performances « du marché ». Le portage sur ces plates-formes pose alors certains problèmes :

- ✓ Logiciel trop lourd et donc nécessitant trop de puissance de calcul
- ✓ Logiciel trop gros et donc nécessitant trop de mémoire
- ✓ Outils de développement inadaptés ou voire inexistants
- ✓ Librairies et OS inexistants pour certaines plateformes

La scalabilité que l'on pourrait appeler « inverse » à celle d'augmenter les ressources n'est donc pas une chose simple pour les développeurs. Elle est pourtant nécessaire pour offrir aux utilisateurs des plates-formes efficientes énergiquement et étant aussi moins polluantes à fabriquer. Les solutions logicielles existantes nécessitent donc d'être adaptées. L'idéal pour les développeurs de solution est de prendre en compte ces plates-formes dès la conception. Des projets existent comme ARM Linux mais sont cependant encore trop peu nombreux. Les développements se focalisent effectivement sur les processeurs les plus répandus (les derniers Intel et AMD), et dans ce cas les développeurs doivent eux même optimiser les logiciels.

Optimisation des couches basses

Introduction

On appelle les couches basses du logiciel l'ensemble des modules logiciels qui permettent d'assurer le fonctionnement du système et l'interface avec le matériel. On considère généralement deux parties : le noyau qui gère le système de fichier, les processus, la mémoire et les drivers qui gèrent les parties physiques (périphériques par exemple). Cette architecture permet d'une part au noyau (ou système d'exploitation) de s'abstraire de la connaissance des différentes périphériques et aux logiciels applicatifs de ne pas gérer la complexité du matériel. Les drivers et les noyaux diffèrent grandement des logiciels applicatifs : Gestion par interruption des drivers, vitesse de gestion des périphériques plus lente que les cycles processeurs… Le développement des drivers est donc plus contraint que celui des applications. De plus Les drivers sont communs à toutes les applications et nécessitent donc d'être performant. L'éco-conception d'un driver doit donc se focaliser à rendre le logiciel le plus efficient possible. Sans cela, l'impact des couches basses pourra être très négatif.

Connaissance du matériel

La connaissance du matériel est importante pour le développeur. Comme l'ingénieur qui conçoit un pont et qui doit connaître la résistance des matériaux, le développeur doit savoir comment le matériel se comporte. Sans cela les green patterns n'auront aucun sens et risqueront de ne pas atteindre le but voulu.

Des connaissances de base seront suffisantes dans la plupart des cas :

- ✓ Architecture matériel d'un PC
- ✓ Mécanisme de gestion de l'énergie du processeur
- ✓ Mécanisme de gestion de la mémoire

Connaître par exemple un périphérique très consommateur permettra de se focaliser sur des mécanismes optimisant l'utilisation de celui-ci. Ci-dessous par exemple la répartition de la consommation pour les périphériques d'un smartphone. Il est clair qu'une optimisation de la gestion de l'affichage sera plus bénéfique que celle de la RAM.

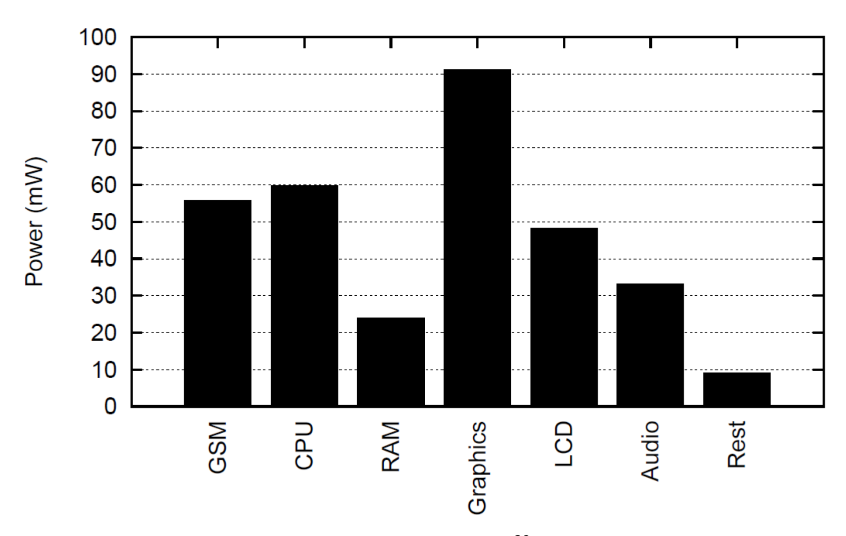


Figure 20 - Consommation d'un smartphone[32]

Cette connaissance ne s'arrête pas simplement à l'architecture matérielle d'une plate-forme mais aussi à l'architecture et au comportement Internet. Un développeur travaillant sur une application répartie sur internet doit avoir des notions sur le routage, les routeurs… pour analyser l'impact de son logiciel et mieux l'architecturer. Ceci est cependant de moins en moins facile compte tenu des évolutions technologiques.

Les développeurs ne peuvent plus maîtriser une multitude de langages et de technologies. Les technologies multi-cœurs en sont la preuve. La complexité pour atteindre la performance maximum donnée par la loi d'Amdhal (Voir chapitre « Évolution du logiciel » p.40) montre très bien cette problématique. Pour pallier cela, le développeur peut utiliser des outils et des framework prenant en compte cette complexité. On peut citer pour la parallélisation Intel Threading Building Blocks ou OpenMP. Ces outils arrivent souvent avec trop de retard par

[32] An Analysis of Power Consumption in a smartphone – Aaron Caroll – NICTA and university of New Wales

rapport aux les technologies et laissent les développeurs tâtonner dans leur coin !

D'autres éléments ne seront cependant pas directement maîtrisables par le développeur et nécessiteront une attention particulière. La compilation, les librairies et le langage sont, dans l'optique de l'optimisation matérielle des éléments à traiter. Les options de compilation vont permettre de rendre le logiciel le plus efficient possible en consommation de ressource ou de mémoire (Voir chapitre « Efficacité de calcul » p.168 pour plus de détail). Pour le langage, au-delà de la performance du langage (Voir chapitre suivant), la spécificité des drivers nécessite d'utiliser des langages répondant à certaines contraintes (Par exemple la gestion de la mémoire et du placement des éléments). Le langage le plus souvent utilisé est le C. Cependant les développeurs peuvent utiliser des langages se rapprochant comme le C++. Ces pratiques ne sont pas sans danger pour le développement des driver en termes d'efficience.

Pour aller plus loin

Article Microsoft - C++ for Kernel Mode Drivers: Pros and Cons - http://msdn.microsoft.com/en-us/windows/hardware/gg487420

Retour d'expérience : Meta IT

La start-up bordelaise Meta-IT a conçu un ordinateur de bureau à très haute valeur ajoutée environnementale, recentré sur les besoins essentiels des utilisateurs en entreprise et collectivité : web, courriel, suite bureautique, ERP, applications métiers. Son nom est ALT.

C'est un produit simple, tout intégré, économe en énergie, totalement silencieux, utilisant des matériaux recyclables, «

Made in France », sans emballage, à longue durée de vie. Il fonctionne sous GNU/Linux ou Microsoft Windows.

Parmi ses spécificités : peu de pièces mécaniques pour allonger sa durée de vie, une alimentation électrique spécifique de 30 W qui délivre un rendement de 90%, un assemblage en France à Talence et Bidart dans les Pyrénées-Atlantiques.

Côté caractéristiques techniques, ALT repose sur un processeur Intel Atom Z530 (1.6 GHz), chipset Intel US15, 1Go de RAM (DDR2), et une carte Compact Flash ou un SSD (disque dur flash) pour le stockage. La dalle LCD et l'alimentation sont prévues pour fonctionner une dizaine d'années. Le fabricant propose une extension de garantie jusqu'à 5 ans.

Le respect de ces caractéristiques a obligé Meta-IT à lancer des actions concrètes visant à optimiser les logiciels et couches-basses GNU/Linux de ALT et sur les serveurs.

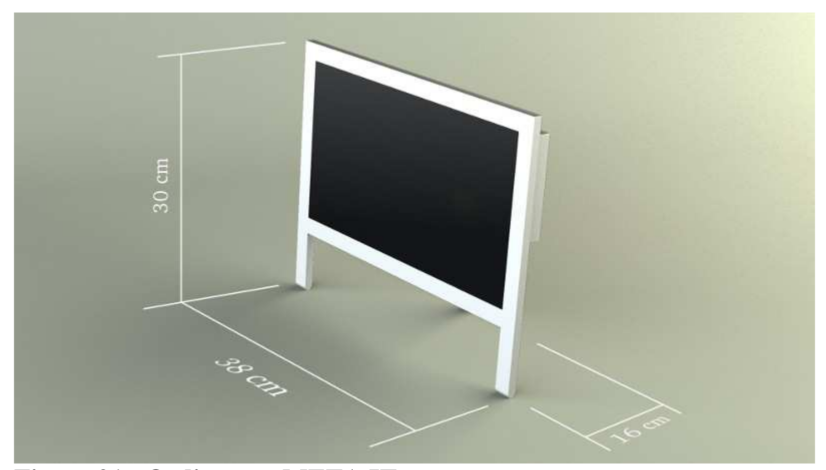


Figure 21 - Ordinateur META IT

Voici le témoignage d'Oliver Cortes, Responsable du logiciel chez Meta-IT

“ Nos actions, dans ce que j'appelle l'éco-développement logiciel, ont porté notamment sur les points suivants :
- réduction du temps de démarrage des machines, avec pour ce faire une analyse du process de démarrage, et plusieurs essais de configuration de l'image de démarrage (j'ai encore dans mon cahier les différentes configurations essayées, et les chiffres recueillis). Une fois la bonne configuration trouvée, nous avons épuré les scripts de démarrage en supprimant les « inutiles ». Cette opération est somme toute classique pour un informaticien, mais nous l'avons reproduite à l'échelle industrielle puisque nos machines sont installées en atelier à travers une procédure automatique

Pour la création et le développement de notre solution de gestion système (concurrent direct d'active directory), je me suis penché sur l'existant bas-niveau pour implémenter un système original de permissions afin de faciliter le partage de données entre les utilisateurs. Dans les couches basses, nous n’utilisons rien d'autre que des choses existantes depuis de nombreuses années (permissions POSIX et ACLs) dans le noyau GNU/Linux, mais dont personne n'a réellement la maîtrise (aucune utilisation industrielle connue) car ce sont des concepts assez complexes. Contrairement à d'autres solutions qui ont implémenté des interfaces ET des systèmes de gestion de permissions stockés dans des bases de données (donc réinventant un système à côté de celui déjà en place, occupant ainsi plus de ressources et nécessitant des serveurs de bases de données), nous avons « masqué » la complexité des ACLs à travers des outils de haut-niveau qui les manipulent à la place des utilisateurs, évitant les erreurs (fréquentes) et aboutissant à un système très simple de « partage, responsabilité et invitation » pour chaque groupe de travail qui souhaite mettre des documents en commun. Ce point est particulièrement simple du côté des utilisateurs, et particulièrement difficile à expliquer à un néophyte côté logiciel. Pourtant, le code n'est pas très long et il exploite des choses qui sont présentent dans chaque GNU/Linux installé dans le monde, mais que personne n'utilise.

Nous avons des fonctionnalités propres, liées ou non aux précédentes. Dans le contexte du logiciel libre, nous nous posons la question, à chaque fois que cela est pertinent, de savoir si ce que nous allons faire n'a pas déjà été fait. Nous utilisons donc un certain nombre de bibliothèques existantes dans nos programmes, et nous rassemblons les fonctionnalités afin de diminuer le nombre de programmes. Par exemple, sur nos systèmes multi-utilisateurs, un seul processus commun est en charge d'indexer les données pour les recherches, et chacun des utilisateurs peut interroger une base commune (qui gère les permissions correctement pour les documents privés). Ceci évite d'avoir sur chaque machine, un programme par utilisateur qui indexe et réindexe des choses par ailleurs déjà parcourues. La mutualisation est ici une optimisation à deux niveaux : elle fait gagner des lignes de codes et du temps de maintenance, autant que des ressources CPU et mémoire. Mais en fonction de ce dont nous avons besoin, elle n'est pas toujours indiquée (certaines bibliothèques externes offrent bien plus que ce dont nous avons besoin et leur inclusion peut coûter plus cher en ressource que la ré-implémentation du petit-peu dont nous avons quelque fois besoin).

Sur des points bien précis, comme la surveillance des fichiers, j'ai benchmarké longtemps le logiciel (développé par mes soins), et l'ai ré-implémentée plusieurs fois. L'objectif étant de surveiller un nombre assez conséquent de fichiers (plusieurs centaines de milliers), une solution « par utilisateur » était inenvisageable (bien que ce soit la solution actuellement en place dans le monde…). J'ai fini par choisir une implémentation « couches-basses » externe, sur laquelle j'ai ajouté nos fonctionnalités haut-niveau. Cependant, cette implémentation « bas-niveau » avait une empreinte mémoire bien supérieure à mes précédentes itérations. J'ai donc étudié le code et proposé une modification de quelques lignes qui a permis de diviser cette empreinte mémoire de 35%, et le développeur externe l'a acceptée avec joie. Ce gain a été rendu possible par une très bonne connaissance de la plate-forme et du langage utilisé, ainsi que de l'interpréteur/compilateur. La bonne pratique ici consiste donc simplement à connaître avec précision

l'environnement et toutes les fonctionnalités des outils utilisés, ce qui je vous l'assure est loin d'être quelque chose de généralisé au sein des informaticiens développeurs.

D'une manière générale enfin, l'approche que nous menons dans les développements logiciels est une approche raisonnée, ouverte et socialisante. Elle s'inspire fortement des méthodes agiles et du bon sens. C'est plus une démarche globale que l'application de « patterns » ici ou là dans le code. Le développeur doit avoir connaissance de l'environnement dans lequel va évoluer son programme, pour l'adapter en fonction. Il ne doit pas simplement répondre à un problème isolé par un bout de code, sauf quand c'est la meilleure solution. Et le jugement de la qualité de la solution sur plusieurs axes (performance, consommation de ressources, maintenabilité, etc) est un exercice difficile que j'aurai bien du mal à théoriser, tant il fait appel à une multitude d'informations et parfois à l'expérience (qui est quelque fois un ennemi…). Pourtant c'est un exercice que nous tentons de faire chaque fois qu'une décision doit être prise dans la solution META, et c'est majoritairement cela que j'appelle « éco-développement logiciel. »

Gestion d'énergie

La gestion de l'énergie des plates-formes matérielles et des systèmes d'exploitation est un domaine que doit maîtriser les développeurs et plus particulièrement ceux qui travaillent sur les drivers et les noyaux. L'utilisation des mécanismes implémentés permet de diminuer drastiquement la consommation. Pour les systèmes d'exploitation, il existe une norme (ACPI : Advanced Configuration and Power Interface) qui standardise la gestion de l'alimentation et le passage dans les différents états de veille. Pour le matériel, les technologies dépendent des fondeurs de processeurs.

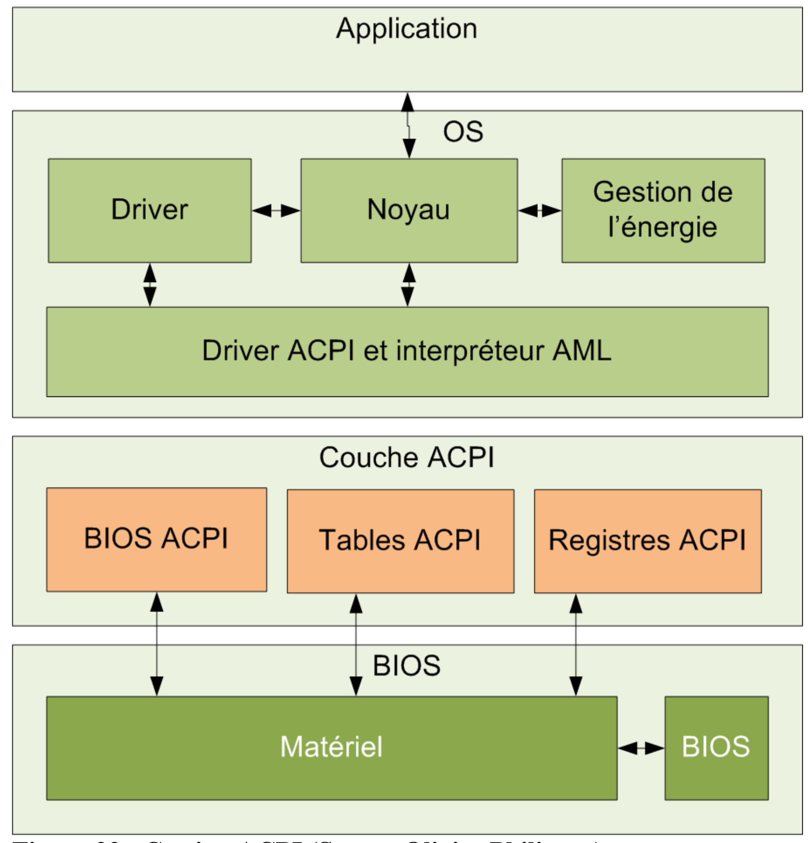


Figure 22 - Gestion ACPI (Source Olivier Philippot)

Les mécanismes généralement implémentés permettent de mettre le processeur dans des états de veille. La puissance de calcul et donc la consommation sont ainsi diminuées. Plus le processeur est dans un état de veille profond, plus le passage dans un état d'activité supérieur sera long. Le fonctionnement est le même pour chaque module. On parlera par exemple de link state pour les périphériques de communication (Ethernet par exemple). Les processeurs Intel définissent par exemple les

états d'activité suivants[33] :

C-state	Max Power Consumption
C0 (busy wait)	35 Watts (TDP)
C1	13.5 Watts
C2	12.9 Watts
C3	7.7 Watts

Figure 23 - Consommation des C-State

Les états du processeur sont les état Cx qui vont de C0 à C3. Dans l'état C0 le processeur fournit des services et donc exécute des instructions. Dans les états suivants, plus aucune instruction n'est exécutée et la consommation du processeur est diminuée. Les mécanismes de diminution de la consommation ne sont pas imposées, c'est uniquement l'état d'activité et les transitions qui sont spécifiées. Le passage entre les états C1 à C3 est géré par des mécanismes différents entre fondeurs. Dans les processeurs Intel et AMD simple cœur par exemple l'état C1 est validé par l'exécution d'une instruction spécifique (Instruction HALT).

33 http://www.lesswatts.org/documentation/silicon-power-mgmnt/#cpupower

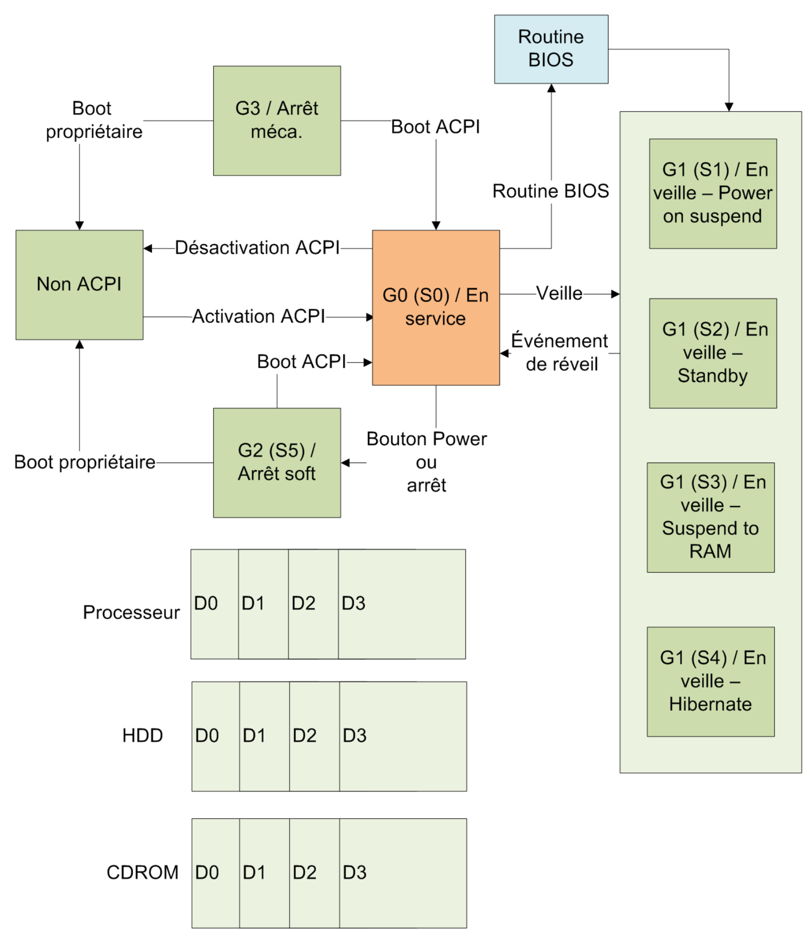


Figure 24 - Gestion ACPI et états (Source Olivier Philippot)

La stratégie sera donc d'éviter toute pratique de développement qui interdise ou réduise le passage dans des états de veille. Le projet LessWatt.org met à disposition des développeurs des bonnes pratiques et des programmes pour réduire la consommation du système GNU/Linux pour les processeurs Intel. Voici les résultats de ces bonnes pratiques sur la consommation du noyau GNU/Linux :

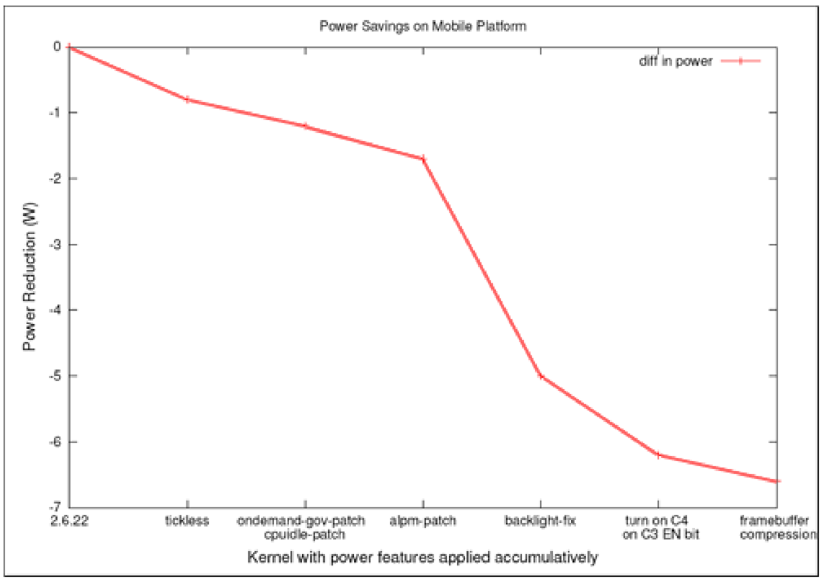


Figure 25 – Gains cumulatifs des bonnes pratiques préconisées par lesswatts.org

Les mécanismes sont spécifiques à chaque processeur, la complexité pour le développeur sera de bien prendre en compte le fonctionnement et l'intégration des fonctionnalités. Sans cela, le logiciel développé n'utilisera aucun des leviers d'économies et fonctionnera dans un mode le plus consommateur. On peut citer les mécanismes suivants :

- ✓ Intel
  - Enhanced Halt State (C1E)
  - Automatic Thermal monitoring
  - Intel Dynamic Acceleration
  - Dynamic FSB Frequency Switching (DFFS)

  - Enhanced Intel Deeper Sleep
  - On-Demand Clock Modulation
  - EIST (Enhanced Intel Speed Technology)

- ✓ AMD
  - Cool'n'Quiet
  - Optimized Power Management (OPM)
  - Dual Dynamic Power Management

Il existe de nombreuses ressources sur internet pour implémenter ces mécanismes : par exemple EIST sur Linux ou les ressources AMD sur AMD Developper central.

Les mécanismes implémentés dans les système d'exploitation permettent d'appliquer une politique de gestion de l'énergie et de faire le lien avec les mécanismes matériels. La norme ACPI définit une partie de ce fonctionnement. Comme pour les mécanismes processeurs, le développeur devra connaître les interfaces de la gestion de l'énergie.

Pour aller plus loin

Microsoft power Management and ACPI http://msdn.microsoft.com/en-us/windows/hardware/gg463220

Projet Lesswatt.org (Intel et GNU/Linux) : http://www.lesswatts.org

Pour GNU/Linux, chaque distribution documente sa gestion de l’énergie. Red Hat

Retour d’expérience : OLPC, la consommation d'énergie comme enjeu

Le projet OLPC a pour objectif de fournir un ordinateur comme outil éducatif aux enfants les plus pauvres de la planète. Pour cela, la fondation OLPC a conçu un ordinateur spécifique, le XO, qui est l'ancêtre des Netbooks.



Figure 26 - L'ordinateur XO

L'ordinateur XO est d'une couleur verte très « flashy » ce qui rappelle la volonté de la fondation OLPC de proposer un produit éco-conçu. Le cahier des charges intègre en effet les 3 piliers du développement durable:

- ✓ Social car l'ordinateur cible les enfants des pays en voie de développement dans l'objectif d'accéder à une meilleur éducation (un des objectifs du millénaire pour l'ONU),

- ✓ Ecologique car le XO limite l'utilisation de matières polluantes, qu'il est conçu pour avoir une durée de vie de 5 ans et que sa consommation d'énergie a été réduite au maximum comme nous allons le voir ci-dessous,

- ✓ Economique puisqu'il s'adresse à des pays qui par définition ont peu de ressources et que son coût de production est donc prévu pour être le plus faible possible (prévu à 100$ à l'origine).

La conception d'un tel outil, initiée par Nicholas Negroponte et d'autres chercheurs du MIT à Boston, s'est notamment attachée à limiter la consommation énergétique. En effet, si dans les pays occidentaux, la prise de conscience de la rareté de l'énergie ne fait que commencer, dans les pays où l'ordinateur XO est déployé, l'électricité est encore une denrée rare. Le plus souvent d'ailleurs, dans les pays utilisant le XO, l'école est le premier endroit à disposer de l'électricité.



Figure 27 - Utilisation du XO sur panneau solaire à Nosy Komba, Madagascar

La recherche d'économie énergétique menée par les équipes du projet OLPC a été un travail à la fois matériel (choisir des composants de faible consommation) et logiciel (piloter efficacement la consommation de ces composants).

Sur le plan matériel, le choix des composants s'est fait au niveau macro (conception d'un écran bi-mode pouvant fonctionner sans rétro-éclairage, choix d'un processeur AMD Geode ayant une fréquence de fonctionnant faible, disque

NAND flash, ...) mais aussi dans le détail (port PS/2 alimenté en 3.3v seulement car il n'est pas prévu de pouvoir alimenter un clavier externe).

Le pilotage consiste pour le logiciel à être capable de réduire la consommation ou de stopper un composant du système lorsqu'il n'est pas utile. Pour cela un module de gestion d'énergie spécifique appelé OHM a été développé au-dessus de la distribution GNU/Linux Fedora utilisée sur le XO. Il ne s'appuie pas sur ACPI pour des raisons d'optimisation. ACPI est par définition un standard permettant de gérer des équipements très différents, son implémentation est donc très générique. Il a été préféré la réécriture d'un système dédié aux composants spécifiques se trouvant dans l'ordinateur XO. Cela a permis de réduire la complexité et la taille du logiciel de pilotage.

Quelques exemples de fonctionnalités intégrées dans la gestion d'énergie du XO:

- ✓ Le système baisse la fréquence de rafraîchissement de l'écran à 25hz (au lieu de 50hz) lorsque le server X Window n'a pas été sollicité pendant 20s.

- ✓ Le système réduit le rétro-éclairage lorsque le server X Window n'a pas été sollicité pendant 1 minute.

- ✓ Lorsque l'ordinateur est en mode « e-book » (clavier derrière l'écran), la CPU est suspendue tant qu'un bouton de contrôle n'est pas appuyé.

- ✓ L'écran et son contrôleur peuvent fonctionner alors que la CPU est suspendue. Le contrôleur gère alors uniquement le rafraîchissement régulier de l'écran.

- ✓ La vitesse de passage en mode veille de la CPU est de 160ms ce qui permet de mettre en veille la CPU entre deux frappes claviers.

- ✓ Le composant gérant le WiFi est capable de fonctionner de manière isolée. Comme l'ordinateur XO propose un WiFi mesh où chaque XO sert automatiquement de relai WiFi aux autres XO à proximité, le composant fonctionne même lorsque l'ordinateur est en veille. Il fait alors uniquement du routage de trame et « réveille » le processeur lorsqu'une trame nécessite un traitement.

Une autre particularité du XO est l'interface graphique Sugar. Sugar propose un bureau et un lanceur d'applications (appelées « Activités ») simplifiée. Le fenêtrage dans Sugar est très limité, une seule application est utilisable et active à la fois. Cette absence de réel multitâche évite de charger intensément la mémoire et la CPU de l'ordinateur . Il évite également d'avoir des applications qui tournent en tâche de fond et qui sont souvent une source de déperdition d'énergie (c'est une des raisons principale pour laquelle iOS et Windows Phone 7 ne proposaient pas de multitâches à leur lancement). A défaut de faire plusieurs applications en même temps, Sugar propose par contre simplement, en utilisant le WiFi de partager une application à plusieurs (un peu comme Google App).

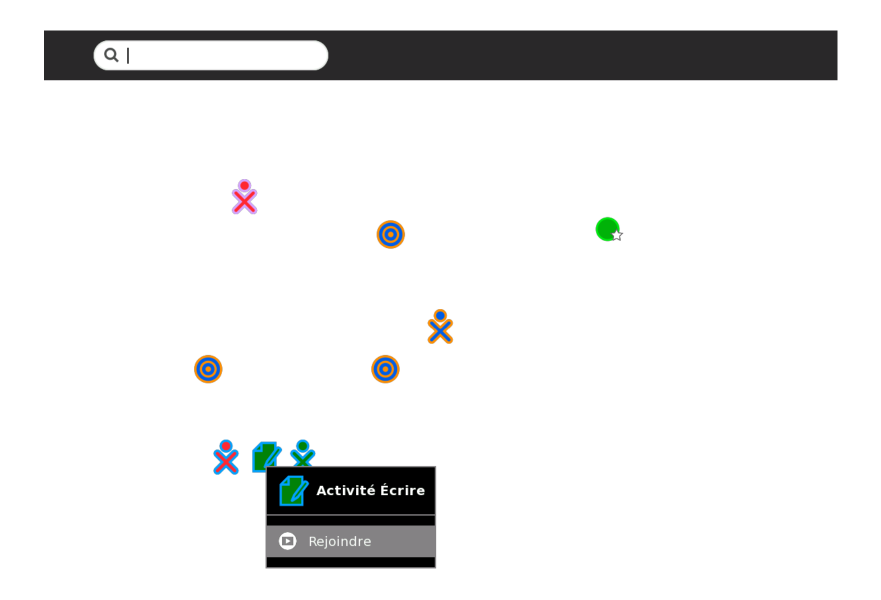


Figure 28 - L'écran Sugar visualisant le partage d'applications

Au final l'ensemble de ces éléments permet aujourd'hui au XO d'avoir une consommation moyenne de 5 Watt est d'à peine de 1 Watt lorsque l'ordinateur est en veille avec le WiFi actif. C'est 10 fois moins qu'un ordinateur portable standard !

Pour aller plus loin

http://laptop.org

http://olpc-france.org

Langages

Langage interprété, langage machine

Les langages peuvent se caractériser par la façon dont ils sont présentés à la machine. On peut diviser pour simplifier les langages en deux grandes catégories : les langages machines et les langages interprétés. Les premiers sont des langages qui sont compris par les processeurs, c'est-à-dire qu'ils sont écrits avec des instructions compréhensibles par le processeur. On peut lister dans cette catégorie l'assembleur, le C… Ces langages offrent cependant une abstraction pour le développeur car la programmation en instruction machine est assez compliquée (par exemple des manipulations de registre, des opérations sur les données binaires…). Les compilateurs réécrivent le code dans un langage compréhensible par le processeur lors de l'exécution. Les langages interprétés sont quant à eux des langages non compréhensibles par le processeur. Il faut donc un programme qui lors de l'exécution « interprète » le langage en des instructions machines. Dans cette catégorie on peut lister php, les langages de scripting comme Perl… Il existe des langages intermédiaires appelés semi-interprétés qui sont transformés par un compilateur. Le code intermédiaire fonctionne sur des interpréteurs ou des machines virtuelles plus complexes. Parmi ces langages on peut noter Java

La différence d'impact sur la consommation d'une application se fera donc sur l'optimisation plus ou moins forte du langage par rapport au langage de la machine. Un langage machine sera très proche du processeur et donc lors de l'exécution, cela demandera moins d'effort du processeur pour réaliser les instructions. Un langage interprété demandera plus de ressource au processeur à la fois pour faire tourner la machine d'interprétation et pour interpréter le langage. Cependant les langages interprétés gardent l'avantage de la simplicité et surtout de la portabilité sur différentes machines. De plus,

certaines architectures et techniques permettent d'avoir des interpréteurs assez performants. Le choix entre les langages doit donc se faire en analysant clairement la performance de la technologie et surtout en effectuant une analyse du cycle de vie du logiciel.

Durabilité des langages

Les langages évoluent, certains apparaissent, d'autres disparaissent…. Le choix d'un langage doit prendre en compte ce critère car le développement d'un logiciel dans un langage obsolète ne serait par définition pas durable. Si un langage devient obsolète, d'une part les outils et les ressources associées seront plus rares, et d'autre part les compétences pour maintenir le logiciel ne pourront plus être trouvées. Sauriez-vous maintenir un logiciel rédigé en Cobol il y a 20 ans ? On peut également évaluer la capacité des langages à évoluer vers les nouvelles architectures (passage du 32 au 64 bits par exemple). Foxpro de Microsoft, dont l'arrêt du support est prévu en 2015, en est l'exemple.

L'obsolescence d'un langage est une chose compliquée. Cela nécessite une veille importante pour identifier cette caractéristique. Voici quelques pistes :

- ✓ Aidez-vous des classements tels que celui de Tiobe[34] Ils vous indiquent la popularité des langages. Mais attention, ce n'est qu'une popularité et cela n'indique pas la durabilité. Mais c'est un bon indicateur !

- ✓ L'âge du langage est un indicateur important. S'il est associé à une popularité, cela peut indiquer que le langage est toujours actif. Un nouveau langage pourra être risqué car rien n'indiquera qu'il sera adopté par les développeurs.

- ✓ Il faut faire attention au leader du langage. Un langage lancé

---

[34] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

par une société et non par une communauté pourra par exemple être un point d’attention important. La dépendance à la société créatrice sera en effet un risque (objectivité, durabilité…)

Position Nov 2011	Position Nov 2010	Delta	Programming Language
1	1		Java
2	2		C
3	3		C++
4	5		C#
5	4		PHP
6	8		Objective-C
7	7		(Visual) Basic
8	6		Python
9	11		JavaScript
10	9		Perl
11	10		Ruby
12	20		PL/SQL
13	13		Lisp
14	15		Pascal
15	21		MATLAB
16	12		Delphi/Object Pascal
17	23		ABAP
18	22		Lua
19	16		Ada
20	19		RPG (OS/400)

Figure 29 - Classement Tiobe Novembre 2011

Exemple de projet impacté par le choix du langage

Il existe dans le monde logiciel des exemples de sociétés qui ont été impactées par le choix d'un langage et qui ont dû re-développer le code dans un autre langage pour des raisons de performance. Les méthodes peuvent être variées pour optimiser la performance :

- ✓ compiler le code juste avant l'exécution. Il s'agit alors d'une compilation juste à temps (JIT). On peut citer C# ou Python avec PyPy
- ✓ Semi-compiler le code comme pour Ruby avec YARV
- ✓ Compiler en code natif comme PHP avec HipHop

Facebook avec le projet Hiphop[35] est l'exemple le plus célèbre de travail sur l'optimisation d'un langage. Pour des raisons de performance et d'optimisation de l'occupation de ses serveurs, Facebook a mis en place un interpréteur permettant de transformer le code PHP en code C++ compilable en code machine. Le compilateur ainsi obtenu à été mis à disposition par Facebook à la communauté sous licence PHP.

On peut aussi noter le logiciel Evernote[36] qui est passé au C++ pour éviter une obésité croissante.

Choix du langage

L'impact d'un logiciel sur la consommation du processeur est variant en fonction du langage. Le choix d'un langage doit donc prendre en compte cet impact. Le lancement d'un projet logiciel doit être fait en estimant les futurs impacts. La simplicité des

[35] Source :http://developers.facebook.com/blog/post/358

[36] http://blog.evernote.com/2010/10/26/evernote-4-for-windows-is-here/

langages interprétés ou le buzz autour de certains langages ne doit pas être le critère unique de choix. Ce choix sera d'autant plus important que le logiciel se destiné à un large déploiement. Ces choix sont d'autant plus négligés que les développeurs ont généralement 1 à 2 langages de prédilection et d'expertise. Plus le nombre de langages pris en compte sera grand, plus le choix du langage permettra une optimisation de l'impact environnemental.

Voici quelques pistes de choix :

- ✓ L'impact d'un langage sur la consommation est d'autant plus important que le logiciel est déployée à grande échelle. La criticité de l'impact du langage sur la consommation dans le choix doit donc être élevé avec l'échelle de déploiement (et donc le choix doit aller plus vers des langages machines)

- ✓ Pour des logiciels n'étant voués qu'à une exploitation unique ou faible, un temps de développement important pourrait être plus impactant (compte tenu de l'impact de la phase de développement). Le choix d'un langage induisant une prise en main et un développement simple doit donc être privilégié (langage interprété type scripting par exemple)

- ✓ Dans le cas d'un logiciel destiné à être porté sur différentes machines, le choix du langage peut se faire soit sur un langage machine en prenant en compte la problématique de la portabilité dans l'architecture soit sur un langage interprété si le critère de l'échelle de déploiement le permet.

- ✓ La pérennité du langage doit être prise en compte. Il faut assurer d'une part que les outils seront disponibles tout au long de la vie du logiciel et que les compétences seront aussi existantes.

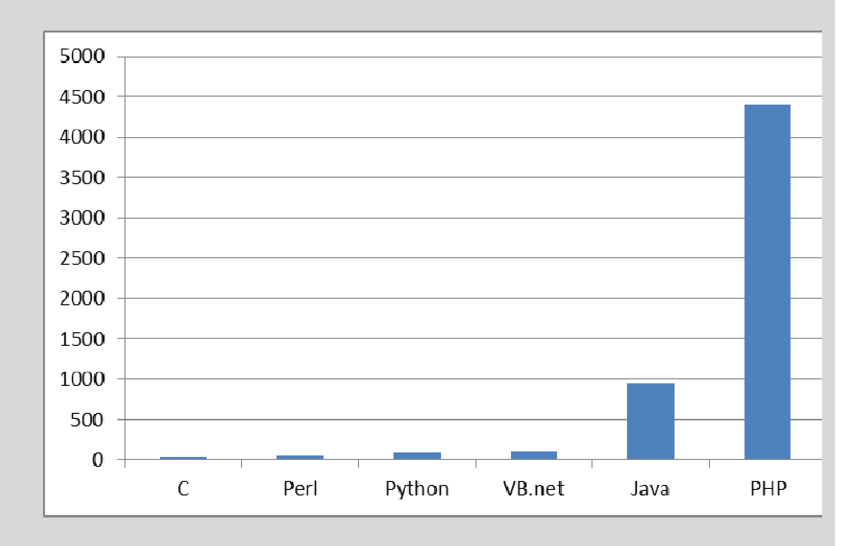
Question…

Des outils de codage automatiques arrivent sur le marché et permettent de générer le code à partir de modèle. Green ? Oui et non.
Oui car cela permet d'une part de simplifier le processus et de modifier rapidement les fonctionnalités et d'autre part d'intégrer des green patterns.
Non car la charge de développement n'est pas totalement supprimée, il faut modéliser pour générer et cela prend du temps. De plus si l'outil n'intègre pas de green patterns ou d'optimisation dans le code, le logiciel généré ne sera pas vert.

Expérimentation Green Code Lab

Vous trouverez dans le code lab un projet d'évaluation des langages. L'algorithme de hachage cryptographique SHA (Secure Hash Algorithm) a été codé dans différents langages. Les langages suivants ont été implémentés : C, Javascript, Perl, PHP, Python, VB.Net

Les résultats des temps d'exécution en ms sont les suivants :



Il y a donc des différences non négligeables. Attention cependant aux conclusions car l'optimisation lors de la compilation peut changer de façon importante les résultats.
Les travaux à réaliser sont maintenant de voir l'impact des optimisations et de faire des mesures de consommation

GREEN PATTERNS

Efficacité d'architecture

Introduction

Une mauvaise architecture logicielle c'est comme une mauvaise architecture de bâtiment : C'est bancal, ça fuit et on peut difficilement faire des extensions. L'architecture du logiciel est une phase sur laquelle on doit particulièrement réfléchir. Durant cette phase, des regroupements de code vont être faits et les interactions entre ces groupes vont être décrites. Ce travail va permettre de décrire précisément la structure du futur logiciel et comment est composé le logiciel (contrairement aux spécifications qui diront ce que l'on doit faire). L'architecture logicielle est devenue très importante dans un projet logiciel. Ceci est dû aux contraintes de complexité de code de plus en plus importantes et aux besoins de qualité grandissants.

Le cerveau humain à partir d'une certaine complexité peut difficilement gérer certains problèmes. C'est le cas de la structure d'un code logiciel. Pour un petit programme, le développeur pourra par lui-même optimiser la structure du code et donc augmenter la performance de son application. A l'opposé, pour un code complexe ou développé par plusieurs personnes, l'optimisation sera beaucoup plus compliquée. Certes le résultat du code sera fonctionnel mais il utilisera beaucoup plus de ressource que nécessaire. Le code se rapprochera sûrement d'un code « spaghetti » où il est impossible de suivre clairement le séquencement d'un traitement d'une donnée.

En éco-conception, comme dans d'autre domaine, ce qu'il faut retenir sur l'architecture c'est : la simplicité, la disponibilité et la sûreté de fonctionnement, l'adaptabilité et l'évolutivité. Il est important de réaliser une phase d'architecture, même pour un

programme simple. Lors de cette phase, en plus des critères de conception classique d'une architecture, on surveillera les aspects suivants :

- ✓ Efficacité de répartition
- ✓ Efficience de répartition
- ✓ Abstraction des couches de calcul
- ✓ Modularité de l'architecture

Pour aller plus loin

Architecture logicielle : Concevoir des applications simples, sûres et adaptables - Jacques Printz – Editions DUNOD

Efficacité d'unité

Pour éviter l'effet spaghetti, le logiciel devra être découpé en unités cohérentes. Ce découpage permettra de réaliser des traitements avec une certaine cohérence de temps et d'espace. Cette cohérence sera plus simple à gérer par le système, les données pourront être par exemple calculées dans une même fenêtre temporelle (interruption, tâche…) ou dans le même espace mémoire. De plus certaines données, comme celles issus des traitements intermédiaires, ne sortiront pas de cette unité, et ne pollueront donc pas le reste du code. L'interface entre les unités se résumera à des données importantes ou nécessaires à la communication entre modules.

De nombreuses techniques d'architectures et design patterns existent et permettent d'atteindre ce résultat. On peut citer :

- ✓ MVC (Modèle Vue Controleur) : l'objectif est de séparer les données (model), de représenter des objets manipulables (view) et de contrôler ces données.

- ✓ Multi-tier : le concept de couche logicielle permet une flexibilité et une réutilisation des unités logicielles : par exemple présentation, application et gestion des données.

- ✓ Peer-to-peer : Architecture de calcul réparti où chaque nœud a des priorités équivalentes aux autres.

Ces techniques sont bien-sûr des cadres de pensée qui permettent de concevoir une architecture, il faut intégrer dans ce cadre des contraintes d'efficacité (Quelle répartition me permet d'avoir un algorithme le plus performant ?) Ceci permettra de faire des choix qui seront impactants pour la performance de l'application. Globalement, la bonne pratique est de se baser sur les modèles existants et éprouvés plutôt que de ne pas utiliser de pattern. Cela permettra d'avoir une division en unité la plus efficace possible.

Efficacité de répartition

La répartition est la manière de distribuer les unités logicielles sur différentes éléments physiques. La division en unités et la répartition sont donc des activités différentes mais complémentaires. Une efficacité doit donc être associée à l'unité comme à la répartition. Pour l'efficacité d'unité, on se préoccupe de réaliser une architecture qui sera plus rapide et qui consommera moins. Cependant il faut réfléchir à la façon de répartir les différents composants de cette architecture. Prenons le cas d'une architecture logicielle avec des calculs répartis en client/serveur : si on parle en termes d'efficacité, on va se préoccuper de mettre en place une architecture performante coté serveur et coté PC successivement. Cependant, contrairement à un PC où la consommation est constante sur toute la plateforme, la consommation entre le serveur et le client n'est pas similaire. En effet, même si un

serveur consomme beaucoup plus, cette consommation est mutualisée contrairement à un PC. Il est donc plus intéressant dans une application client/serveur de maximiser le nombre de traitement côté serveur. Cependant si l'on prend la bonne pratique précédente (efficacité d'unité), si l'on répartit mal les calculs alors un nombre trop important d'échanges entre le client et le serveur pourra créer une consommation beaucoup plus importante que le gain de déplacement des unités côté serveur.

Exemple : Cas d'application USI green pattern dans le chapitre « Applications Web » p.244

Abstraction des couches basses

L'architecture doit aussi tenir compte de certains niveaux de fonctionnalité et d'une organisation précise entre eux. Chaque niveau de fonctionnalité forme une couche logicielle et doit normalement s'abstraire des couches inférieures. C'est-à-dire qu'une couche ne doit pas connaître toutes les données des couches inférieures. Cela permet d'éviter d'avoir un code « spaghetti » où une couche irait chercher des données dans une couche avec des caractéristiques de fonctionnement totalement différentes des siennes. Ce principe se retrouve de façon naturelle dans celle des drivers. Le driver possède son niveau d'abstraction, il gère des données matérielles; le système d'exploitation lui gère des données issues des drivers ; et enfin les applications logiciels traitent des données offertes par l'OS. Plus on va dans les couches supérieures, plus on s'abstrait du matériel et moins les cadences de traitement sont importantes. En effet, le traitement des ressources matérielles est coûteux : interruptions, gestion des registres, nécessité de traitement rapide. Il est nécessaire pour un microprocesseur de traiter des informations à la microseconde, pour le driver à la milliseconde et pour l'application à quelques dizaines de millisecondes. L'avantage si on applique ce principe, est qu'en regroupant ces caractéristiques de traitement, on gagne en efficience. On évite de plus de perturber les autres couches

logicielles (comme par exemple lorsqu'un logiciel bloque les timers de mise en veille de la gestion de l'énergie). Le choix des niveaux d'abstraction se fera donc en respectant certains principes :

- ✓ Traiter les ressources matérielles dans les niveaux les plus bas

- ✓ Echanger les données uniquement avec les couches directement connexes

- ✓ Traiter les données systèmes et utilisateurs dans les couches les plus hautes

Le gain supplémentaire pour cette pratique sera que le logiciel sera plus lisible et plus maintenable. De plus, il sera portable assez facilement sur différentes plates-formes matérielles. En effet, si l'on s'abstrait de toute notion matérielle, le portage du code sera grandement facilité. On gagnera donc en longévité (voir chapitre « Maintenance » p.232)

Modularité

L'architecture du logiciel doit tenir compte de la modularité d'installation. De nombreux logiciels occupent plusieurs centaines de Mo sur les disques alors que l'utilisateur n'utilise que quelques fonctionnalités. Il est donc nécessaire d'offrir à l'utilisateur la possibilité de n'installer que ce qu'il souhaite. Mais ceci demande d'avoir conçu le logiciel en fonction de cette contrainte. Ce fonctionnement se retrouve dans certains gros logiciels (comme Microsoft Office) mais est encore trop complexe pour l'utilisateur. La preuve est que cette modularité n'est offerte que dans le mode avancé d'installation.

Dans des logiciels comme Firefox ou Chrome on trouve ce type de modularité mais avec des modules qui n'appartiennent pas à l'éditeur : on peut ajouter des modules complémentaires développés par d'autres personnes que le fournisseur du

logiciel. Ceci pose cependant la question de la qualité des modules supplémentaires : sont-ils vraiment compatibles ? Quel est leur performance ? Voir chapitre « Guideline » p.235 pour ce sujet.

Efficacité de calcul

Introduction

L'efficacité d'un processus ou d'un travail est la caractéristique à effectuer la même tâche avec moins de ressource. Dans le cas d'un calcul informatique la ressource utilisée est le processeur. Elle peut être caractérisée par différentes notions : temps processeur, nombre de ticks d'horloge, charge CPU… mais dans tous les cas, elle représente une information sur la quantité de ressource du processeur utilisée par le calcul. En utilisant moins de ressource CPU, la tâche sera réalisée en moins de temps et consommera donc moins d'énergie. Le deuxième effet bénéfique sera que les ressources du processeur seront libérées plus tôt soit pour effectuer un autre calcul, soit pour passer en veille.

L'amélioration de l'efficacité d'un calcul se fait en améliorant l'algorithme de calcul. Généralement un algorithme se compose de plusieurs mécanismes qui peuvent être coûteux en temps. Une boucle qui aura pour objectif de surveiller un appui sur une touche sera par exemple coûteuse car elle monopolisera le processeur. Ce coût devra d'autant plus être réduit qu'il n'apporte pas de vraie valeur ajoutée au programme. En plus de ces algorithmes, le calcul pourra être optimisé en améliorant les chemins. En effet, un calcul permet par une réaction (appui sur une touche par exemple) d'arriver à un certain résultat (l'affichage d'un son dans l'exemple de la touche). Pour arriver à ce résultat, le programme va passer dans différents états (ou points de passage). Comme pour une voiture qui va devoir aller d'un point A à un point B, réduire le nombre de points de passage va permettre de rendre le calcul plus efficace. Cette optimisation est d'autant plus importante que les points de

passage peuvent être coûteux. Un point de passage comme une condition qui va permettre de choisir entre deux chemins consomme un nombre de ticks d'horloge du processeur.

Ce chapitre traite des mécanismes les plus répandus et les plus consommateurs. Chaque algorithme est bien-sûr spécifique. Leur optimisation doit donc faire intervenir ces bonnes pratiques mais aussi un peu de réflexion.

Multi-tâche

Un système informatique classique a pour objectif de faire fonctionner plusieurs programmes en même temps contrairement à un autre système électronique : l'électronique d'une calculatrice va traiter la fonction de calcul uniquement alors que le processeur d'un PC traitera plusieurs applications, l'affichage, le périphérique… Cela demande donc de faire fonctionner plusieurs programmes informatiques en même temps. Le système informatique traite donc chaque programme alternativement dans des périodes de temps suffisamment petites pour que l'utilisateur ne voie pas cette commutation entre programmes. On appelle cette méthode le multi-tâche.

Les mécanismes de threading des OS actuels sont largement optimisés et robustes. Il faut donc s'appuyer sur eux. Il faudra cependant respecter certaines bonnes pratiques :

- ✓ Ne pas interdire les interruptions. Cela permettra à l'OS de préempter (stopper) la tâche pour débuter une autre tâche, et y revenir plus tard.

- ✓ Eviter des fonctions trop « grosses ». Cela permet à l'OS de traiter la fonction dans une fenêtre de traitement unique sans être interrompu par d'autres tâches

- ✓ Effectuer les traitements des entrées du programme d'abord, puis le calcul et ensuite le traitement des sorties. Cela permet de libérer les drivers rapidement, d'effectuer

les calculs en mémoire de façon regrouper et d'appeler les drivers ou les fonctions de sortie plus tard.

Multi-Coeur

Les processeurs multi-cœur sont un moyen d'optimiser l'efficacité d'un traitement. Cependant, la complexité de la parallélisation n'est pas simple et l'effet escompté peut être inversé. Voici quelques pistes pour optimiser le traitement parallèle. Premièrement, plusieurs modèles de traitement sont possibles :

- ✓ Modèle de décomposition par domaine
  - Décomposition en domaines de données : la donnée est séparée en partie et traitée individuellement
  - Décomposition en domaines fonctionnels : chaque tâche travaille sur des fonctions ou des parties de code séparées
- ✓ Modèle de répartition de traitement
  - Traitement équilibré : Chaque tâche de traitement est équilibrée
  - Traitement déséquilibré : Il y a des différences de charge entre les tâches de traitement

Les consommations sont très proches entre les deux modèle. Cependant il y a des avantages pour le multithreading équilibré : il y a une division du temps de traitement (En théorie de 2) et le temps gagné permet de passer en mode idle[37]. Il faut équilibrer le traitement car le traitement peu prendre plus de temps pour se terminer et cela va monopoliser une

---

[37] http://software.intel.com/en-us/articles/creating-energy-efficient-software-part-1/

ressource.

Exemple d'un moteur de traitement graphique

✓ Calcul physique (collision, trajectoire…)

✓ Affichage (rendering…)

✓ Balancé : les objets graphiques divisés en deux et traités de chaque côté (Calcul physique + Affichage)

✓ Déséquilibré : une tâche traite le calcul et l'autre l'affichage

Voici quelques exemples d'outils permettant de développer en multi-cœur :

✓ Intel® Thread Checker

✓ Intel® Thread Profiler

✓ Intel® Thread Building Blocks

✓ Intel® VTune™ Performance Analyzer

✓ AMD® Accelerated Parrallel Processing SDK

Timers Périodiques

Les timers périodiques sont consommateurs en ressources car ils font intervenir de nombreuses ressources du processeur et créent des interruptions qui peuvent perturber les programmes en cours. Ils sont d'autant plus nuisibles qu'ils sont activés généralement tout le temps. Il faut donc faire attention à désactiver les timers s'ils ne sont non utilisés.

L'impact d'une tâche périodique est d'autant plus important que cette tâche ne va pas permettre au processeur de rentrer dans

un état de veille. Dans un guideline, Intel[38] explique que pour une même utilisation CPU, une tâche avec une fréquence d'exécution plus importante et une durée d'exécution plus courte à chaque exécution consommera beaucoup plus de ressource. Il vaut donc mieux augmenter la durée et diminuer la fréquence de la tâche. Selon l'étude, une tâche qui s'exécuterait toutes les 10ms pendant 10us consommerait 20% de plus par rapport au repos alors qu'une tâche qui s'exécuterait toutes les 100ms pendant 100us consommerait 3% en plus par rapport au repos.

Boucles

Les boucles de programme sont des mécanismes très répandus car ils permettent de réaliser de nombreuses opérations simplement. Cependant ils sont très consommateurs en ressources car ils monopolisent le processeur. De plus le nombre de boucles à réaliser est souvent très important et même parfois infini (et cela peut causer des bugs). La simplicité des boucles doit être maniée avec précaution car l'impact de l'implémentation d'une boucle peut être très négatif. Voici quelques moyens d'optimisation :

- ✓ Sortie de la boucle : Il faut prévoir au plus tôt une condition de sortie permettant d'arrêter l'itération. Code d'erreur, actions utilisateurs… Toute condition permettant de sortir de la boucle au plus tôt permettra d'éviter des traitements inutiles.

- ✓ Fin de boucle optimisée : Le nombre d'itérations maximales d'une boucle est souvent choisi sans réflexion d'optimisation. La prise en compte du contexte et du besoin est donc à prendre en compte dans le choix du nombre d'itérations. Pourquoi mettre une boucle de surveillance au maximum possible (255 par exemple) alors que d'autres données

[38] Energy-Efficient Platforms – Considerations for Application Software & Services - http://software.intel.com/file/38273

peuvent être prises en compte (temps maximum de saisie, timeout existant…)

- ✓ L'utilisation de boucles pour surveiller des données est à proscrire. Il faut privilégier l'utilisation d'interruptions ou de multitâche. Dans cette optique, on pourra par exemple mettre en place un microprogramme qui traite de la surveillance d'un périphérique et un autre qui traite les données. Cela permettra ainsi à l'OS de gérer de façon plus flexible l'ensemble de la fonction.

Librairies

L'utilisation de librairies permet d'optimiser le calcul dans certains cas. En effet, les librairies sont généralement des programmes stabilisés et optimisés. Il est donc préférable de s'appuyer sur ces bibliothèques. Le web regorge de librairies mises à disposition. On privilégiera les librairies optimisées pour le matériel.

On peut citer quelques librairies :

- ✓ Intel
    - o Utiliser les librairies telles que MMX pour utiliser au mieux des accélérations du matériel
    - o Utiliser les bibliothèques tel Intel Performance Primitive pour les calculs

Il faut cependant faire attention en termes d'efficacité quant à l'utilisation de bibliothèques. En effet, elles sont conçues de façon générique et intègrent parfois des mécanismes qui ne sont pas nécessaires au programme. Dans ce cas une phase d'optimisation de la librairie sera nécessaire avant l'utilisation. Plus généralement, le choix d'une librairie doit faire l'objet d'un benchmark avec d'autres libraires pour comparer l'efficacité.

Il est possible qu'une solution basée sur des librairies soit en définitive peu efficace (manque dans librairies dans certains domaines, librairies peu stabilisées…). La solution d'un programme à façon sera alors l'unique solution. Dans ce cas, pourquoi ne pas partager ce code avec la communauté pour favoriser la réutilisabilité et donc l'éco-conception collaborative ?

Compilateurs

Comme nous l'avons vu dans le choix du langage, la manière dont le code est interprété impacte l'efficacité. Il en est de même pour la compilation du code (pour les langages non interprétés). Un code qui aura été compilé avec des options optimisées prendra beaucoup moins de ressource qu'un code non optimisé. Ces options varient en fonction du compilateur mais on retrouve généralement les mêmes concepts : optimisée pour la vitesse, optimisée pour l'espace… L'optimisation en vitesse sera toujours plus intéressante en termes d'efficacité et sera à privilégier par rapport à des optimisations en termes d'espace mémoire.

Idées d'action Green Code Lab :

Evaluer les impacts sur la consommation en fonction des optimisations et des types de logiciel.

Le compilateur joue aussi un rôle important. Il faut privilégier un compilateur qui permet de maximiser l''optimisation.

Par exemple, pour le compilateur gcc, on a différents niveaux (-O1…) qui permettent d'agir sur la performance et l'occupation mémoire[39]. On peut de plus préciser les plates-formes matérielles. Les performances peuvent alors être multipliées

[39] http://www.linuxjournal.com/article/7269?page=0,0

jusqu'à 4.

Un logiciel sera donc de préférence diffusé en plusieurs versions en lien étroit avec l'architecture matérielle.

Pour les langages interprétés, l'optimisation se fera en choisissant un format adapté de packaging. La problématique sera différente pour les langages interprétés sur le web car la ressource mémoire sera non négligeable. En effet, l'optimisation de la taille du programme prendra le dessus sur la ressource processeur compte tenu de l'impact du transport sur le web.

Driver

L'efficacité de calcul des drivers est d'autant plus importante qu'ils sont utilisés en continu et par d'autres programmes. Nous verrons l'importance de la prise en compte du contexte dans les chapitres suivants. Pour ce qui est de l'efficacité, les bonnes pratiques précédentes sont applicables. Il faudra prêter une attention particulière à l'optimisation du nombre d'interruptions et aux timers.

Efficacité de données

Constat

Une des causes de l'obésiciel est l'inflation des données que le logiciel doit traiter : données multimédias, bases de données... Nous avons vu dans l'étude de EcoInfo sur Microsoft (Voir chapitre « Évolution du logiciel » p.40) que le besoin en mémoire RAM et ROM avait augmenté de plus de 100 fois en 10 ans. Le constat est similaire dans de nombreuses applications. L'institut IBM TJ Watson Research Center Hawthorne[40] a étudié pendant 10 ans des applications écrites

---

[40] Building Memory-efficient Java Applications / IBM TJ Watson Research Center Hawthorne, NY USA -

en langage java volumineuses et notamment leurs performances mémoire. Les constats sont que concernant l’espace mémoire Heaps (espace mémoire où sont stockées les données dynamiques), il y a une augmentation de 500 Mo à 2-3 Go ces dernières années mais pas nécessairement d'augmentation du nombre d'utilisateurs supportés ou du nombre de fonctions. Dans la plupart des cas :

- ✓ il faut 1 Go de mémoire pour supporter quelques centaines d'utilisateurs
- ✓ Sauver une session utilisateur nécessité 500 Ko
- ✓ il faut 2 Mo pour un index de texte par document
- ✓ Il y a 100 Ko de données temporaires qui sont créées par clic sur le net

Les conséquences sont néfastes selon l’institut sur la scalabilité, la consommation d'énergie et la performance.

L'augmentation de la puissance de calcul des processeurs permet de traiter les données de façon de plus en plus rapide mais il faut relativiser en prenant en compte le coût d'accès à cette information. En effet, même si les technologies actuelles permettent des gains de performance, il y a des goulots d'étranglement. Un des problèmes de performance le plus important est la différence de performance entre mémoire et CPU. La performance des mémoires n'évolue pas au même rythme que celle des processeurs et les architectures informatiques souffrent de cette contrainte. Plusieurs solutions s'offrent alors aux développeurs : Attendre que la loi de Moore nous permettent d'avoir des architectures matérielles encore plus performantes (et plus consommatrices !), attendre que les outils (compilateurs, framework...) résolvent les problèmes, ou

---

http://www.cs.virginia.edu/kim/publicity/pldi09tutorials/memory-efficient-java-tutorial.pdf

bien appliquer dès maintenant des patterns efficients. La dernière solution est la plus raisonnable. Optimiser la gestion des données en mémoire est en effet indispensable pour rendre un logiciel plus efficient. La solution matérielle, même si nécessaire car il faut réfléchir à des architectures plus efficientes, n'est pas une solution pérenne. Optimiser la gestion de la mémoire est donc non seulement nécessaire pour rendre les algorithmes efficients mais aussi pour réduire l'inflation de la capacité mémoire.

Les optimisations peuvent être classées en deux types : des Green Patterns techniques qui sont des patterns agissant sur la manière dont sont traitées les données en mémoire et les Green pattern fonctionnels qui sont des patterns agissant sur la manière dont sont traitées les données dans les logiciels.

Distance aux données (pattern technique)

Un principe d'efficacité est l'amélioration de la distance aux données (principe de localité). Plus la donnée va être éloignée du processeur, plus l'accès à cette donnée sera coûteux en temps et en énergie. Cette distance est une distance virtuelle que l'on peut évaluer en nombre d'instructions du processeur et de temps de latence pour accéder à cette donnée. La distance va dépendre de l'architecture et de la technologie. Accéder à une donnée en cache sera par exemple moins coûteux que l'accès à une donnée sur disques dur.

Voici quelques distances classiques :

- ✓ Registre - 1 cycle - Jusqu'à 10 mots par cycle - CPU
- ✓ Cache L1 - 3 cycles - Jusqu'à 2 mots par cycle - CPU
- ✓ Cache L2 - 10 cycles - 1 mot par cycle - circuit externe
- ✓ Mémoire RAM - ~100 cycles - 0,5 mot par cycle - circuit externe

- ✓ Disque dur - ~ 1 million de cycle - 0,1 mot par cycle - périphérique
- ✓ Bande externe - long - peu de mots par cycle - périphérique
- ✓ Donnée réseau - très long - très peu de mots par cycle - périphérique

L'architecture de la mémoire est donc une des clés pour optimiser l'efficience de la gestion des données. Plus les données seront proches, moins le calcul consommera d'énergie. Il faut donc appliquer des patterns qui permettent d'avoir le plus possible de localité :

- ✓ Travailler le plus possible en cache
- ✓ Regrouper les données et le code en fonction de leurs localité

Attention néanmoins à n’abuser de ce dispositif que pour des données souvent accédés ou souvent modifiées. En effet, la souscription systématique à ce principe peut provoquer au final un prérequis matériel supplémentaire (cash ou RAM) qui conduit à l’obsolescence de l’architecture et vraisemblablement à son upgrade.

Cache (pattern technique)

Certaines pratiques sont nécessaires pour optimiser l'utilisation du cache. Plus les données seront traitées dans le cache, moins le code passera du temps à aller chercher les données plus loin (en RAM par exemple). Il faut donc tenter de rassembler les données fréquemment utilisées dans une unité de temps et d'espace. De plus il faut éviter ce que l'on appelle les pertes de caches (Cache misses). Lorsque le CPU recherche une donnée dans un des caches et qu'elle n'y est pas, alors le processeur recharge dans le cache la donnée.

Cela est coûteux. Les causes de pertes de caches peuvent être classifiées de la manière suivante

- ✓ Pertes obligatoires : c'est la première fois que la donnée est accédée et elle ne se trouve pas dans le cache
- ✓ Pertes dues à la capacité : il n'y a pas assez de place dans le cache pour traiter la donnée
- ✓ Pertes dues à des conflits : plusieurs données sont en conflit pour les mêmes espaces de cache.

Plusieurs techniques sont possibles pour éviter ces pertes. Nous allons lister quelques pratiques, cette liste n'est pas exhaustive. Nous souhaitons ici montrer aux développeurs que certaines pratiques sont possibles. Christer Ericson de Sony classifie ces pratiques en se calquant sur le développement durable et le principe des 3 R[41] :

- ✓ Réarranger : modifier les patterns pour augmenter la localisation spatiale
- ✓ Réduire : des formats plus petits et plus intelligents, de la compression de données
- ✓ Réutilisation : Augmenter la localisation temporelle (et spatiale)

Voici quelques bonnes pratiques :

Réutilisation des données : il faut tenter de regrouper les traitements effectués sur les mêmes données.

---

41 http://www.research.scea.com/research/pdfs/GDC2003_Memory_Optimization_18Mar03.pdf

Voici un exemple de traitement optimisé d'un ensemble de données de 1024 points avec des filtres de différentes tailles[42].

Filter Size T	Cache Efficiency
16	92.4%
32	96.0%
48	97.3%
64	97.8%

Figure 30 - Réutilisation des données

On voit que plus le filtre est grand, plus la fenêtre de convolution est grande et plus il y a de réutilisation de données. L'efficacité est donc améliorée. Cependant l'optimisation du cache ne peut pas être le seul critère de choix de configuration des données. Dans l'exemple, entre 2 filtres de taille différente, le résultat est très différent. Les caractéristiques d'un filtre sont choisies sur des critères de comportement du filtre. Le choix entre plusieurs filtres sera donc fait pour des résultats souhaités comparables. De façon générale les optimisations de la mémoire sont à mettre en balance avec d'autres contraintes (fonctionnelles par exemple).

Pré-fetching : il est possible de pré-charger les données. Généralement, il n'est pas nécessaire d'effectuer cette action. Cependant, dans certains cas il est nécessaire d'indiquer au processeur de pré-charger les données dans le cache, plus particulièrement dans les boucles de traitement (par exemple traitement de pixel dans une image). Exemple : Préchargement de données dans le cache en delphi[43].

Fusionner les tableaux : deux tableaux séparés peuvent être

[42] http://www.eetimes.com/design/signal-processing-dsp/4016948/Optimizing-for-cache-performance-part-2
[43] http://esibert.developpez.com/delphi/prefetch/

en conflit pour le même bloc de cache. Les entrelacer dans une même structure permettra de les référencer ensemble et donc d'éviter une perte[44] :

```
int val[SIZE] ;
int key[SIZE] ;
```

sera transformé en :

```
struct merge {
	int val ;
	int key ;
} ;
Struct merge MA[SIZE]
```

Réorganisation des structures : Stocker les éléments en fonction de leurs accès va permettre d'optimiser la performance. Par exemple en rassemblant les données auxquelles le programme accède de manière concomitante et fréquemment en début de structure. Les autres données (gestion des erreurs, affichage...) sont mises en fin de structure.

Hot/Cold spitting : Il est possible de réduire les pertes en divisant les structures selon la fréquence d'accès : hot (fréquemment accédé) et cold (rarement accédé). Cela permet aux données « hot » de rester plus souvent dans le cache et d'être traitées ensemble

[44] http://www.ece.unm.edu/~jimp/611/slides/chap5_3.html

hot field :

```
Struct S
{
void *key;
S *pNext;
S2 *pCold;
}
```

Cold field

```
Struct S2
{
int count[10];
}
```

En général : les améliorations et optimisations plus générales (et qui sont traitées dans ce chapitre) permettent d'éviter cette perte de cache. En effet, plus la gestion de la mémoire et des données sera performante, plus les données resteront en cache.

Coût des données (pattern technique)

La taille des données n'est pas forcément celle que l'on croit quand on développe. Les contraintes imposées par les architectures matérielles obligent les compilateurs et les environnements d'exécution à ajouter des données aux données utiles et à les réorganiser en mémoire. L'alignement et la taille d'adressage sont par exemple des contraintes « dimensionnantes ».

Les compilateurs doivent suivre les restrictions d'alignement des données définies par les microprocesseurs. Cela signifie que les compilateurs doivent

ajouter des octets de bourrage (padding) entre les données définies par l'utilisateur afin que les restrictions imposées par le microprocesseur ne soient pas violée.

```
struct {
   bool b;
   double d;
   short s;
   int i;
};
```

La structure va donc occuper: 1 (bool) + 7 (padding) + 8 (double) + 2 (short) + 2 (padding) + 4 (int) = 24 bytes. Alors que pour

```
struct {
   double d;
   int i;
   short s;
   bool b;
};
```

La taille sera de 8 (double) + 4 (int) + 2 (short) + 1 (bool) + 1 (padding) = 16 bytes.

Ce surcoût de taille est encore plus important pour les architectures 64 bits. Les applications ont généralement de nombreux objets de petites tailles et des pointeurs et cela oblige à ajouter des bits de bourrage, des données supplémentaires et donc à augmenter la mémoire nécessaire.

Pour aller plus loin

Padding en fonction des plateformes et des compilateurs : http://en.wikibooks.org/wiki/Optimizing_C%2B%2B/Code_optimization/Faster_operations

On retrouve les même contrainte dans les langages comme java avec des machines virtuelles JVM. Selon IBM TJ Watson Research Center, le surcoût des données (data overhead) est entre 50 et 90% :

- ✓ pour un double de 8 octets, JVM impose 16 octets d'overhead. la donnée ne compose donc que 33% des 24 octets.
- ✓ pour un treemap de 100 entrées, le premier overhead est de 48 octets et de 40 pour chaque entrée. La taille est donc 3,9kB
- ✓ Pour TreeMap<Double, Double> (100 entrées), l'overhead atteint 82%

Une implémentation alternative serait 2 doubles et l'overhead ne serait que de 2%. Cela ne permet plus d'avoir les mêmes fonctionnalités mais cela montre bien les gains possibles.

Pour aller plus loin

Building Memory-efficient Java Applications Practices and Challenges / IBM TJ Watson Research Center/ http://www.cs.virginia.edu/kim/publicity/pldi09tutorials/memory-efficient-java-tutorial.pdf

Optimisation des IO (pattern technique)

La gestion des données qui sont sur des périphériques externes comme les disques durs passe par ce que l'on appelle des mécanismes de gestion input/output. La gestion des IO dépendant du périphérique qui est composé d'électronique et de mécanique. Les performances varient donc grandement en fonction de la manière dont on gère les IO. Dans le cas du disque dur, la performance sera dimensionnée par : la vitesse de rotation du disque (RPM), le temps de recherche des données (seek time) et le temps de latence de rotation du disque. La performance d'accès dépendra de plus de l'endroit où se trouvera la donnée sur le disque. Pour ce qui est de la consommation, le temps de mise en route du disque dur (spin up) sera aussi dimensionnant. Exemple de consommation : read/write 2W : Spin Up 4W Idle :1W[45] Les patterns d'optimisation vont donc dépendre de la technologie. Nous allons ici lister quelques patterns applicables pour les disques durs mécaniques mais qui ne seront pas forcément applicables pour les disques flash (SSD).

Gestion Synchrone ou asynchrone

[45] http://software.intel.com/en-us/articles/power-analysis-of-disk-io-methodologies/.

Si une application effectue une opération synchrone, elle doit attendre que l'opération soit finie pour continuer son traitement. Dans le cas d'une opération asynchrone, le traitement IO s'effectue en tâche de fond et cela ne bloque pas le système. La performance est ainsi améliorée car le traitement IO et l'application peuvent tourner ensemble.

Taille des blocs

Pour réduire la consommation, il est nécessaire d'augmenter la taille des blocs mémoire. [46]

Size	CPU_ENERGY (mWH)	DISK_ENERGY (mW)
1b	9658.77	1056.0
8b	1336.18	192.32
1KB	24.93	13.76
4KB	24.05	13.56
8KB	23.27	13.23
16KB	22.46	12.98
32KB	22.49	12.85
64KB	22.50	12.96

Figure 31 - Taille des blocs et consommation

[46] Power Analysis of Disk I/O Methodologies - http://software.intel.com/en-us/articles/power-analysis-of-disk-io-methodologies/

Défragmentation

Il faut éviter de fragmenter les fichiers. Un fichier fragmenté est divisé dans plusieurs clusters qui ne sont pas contigus. Cela nécessite donc plus de temps et d'énergie pour lire le fichier

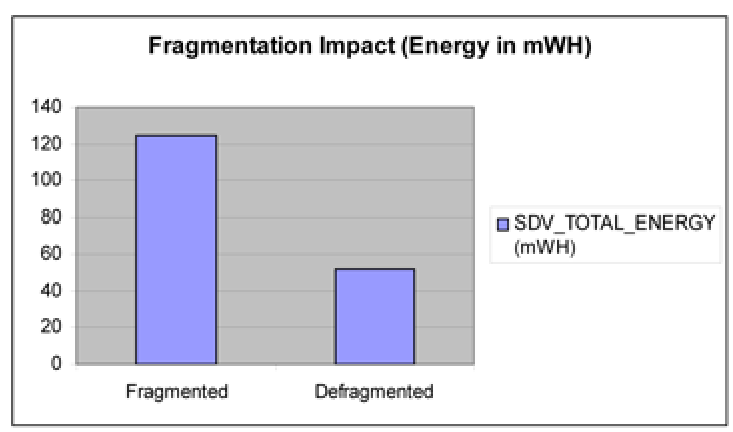


Figure 32 - Impact de la fragmentation

Pour éviter cela, il est préférable dans certains cas de pré-allouer un espace mémoire. Par exemple avec .NET vous pouvez utiliser SetLength().

Green Patterns fonctionnels

En s'abstrayant des concepts matériels vu plus haut, certaines bonnes pratiques sont à appliquer à un haut niveau.

Limiter les accès disques

Les lectures et écritures sur les disques durs doivent être limités. Il faut donc regrouper les traitements de telle manière à ce qu'on laisse le disque au repos le plus possible. Par exemple, traiter et bufferiser en mémoire une partie du fichier et

l'écrire ensuite.

Compresser les données

Compresser les données permet de réduire la taille en mémoire. Le challenge est cependant de mettre en regard le gain en espace mémoire par rapport au coût nécessaire à la compression et la décompression des données.

Regrouper les traitements de données

Comme préconisé dans l'efficacité d'architecture (voir...), il est conseillé de regrouper les modules pour éviter trop de transferts de données. En limitant ainsi les communications, on respecte le principe de localisation temporelle et spatiale des données.

Gérer la durée de vie des données

La création, l'utilisation et la destruction des données doit être gérée correctement. On respectera ainsi le principe de fin de vie des données (Voir chapitre « Fin de vie » p.237).

Utiliser des technologies adaptées à la fonctionnalité

L'utilisation de framework générique ou de certaines technologiques crée très souvent une inflation de mémoire et de performance. Une bonne pratique est donc d'adapter l'environnement à l'utilisation :

- ✓ Pour la mise en place d'une application web distante simple, on utilisera des pages statiques plutôt qu'un CMS. L'espace mémoire sera ainsi drastiquement diminué.

- ✓ Dans le cas d'utilisation de base de données, on pourra utiliser des technologies In-Memory Database

permettant de faire fonctionner la base de données en mémoire plutôt que sur disques durs. On optimisera ainsi le coût d'accès aux données.

Efficacité de prise en compte du contexte

Introduction : de l'île à l'océan

L'île : Le logiciel au commencement

Nous allons simplifier grandement la problématique mais le logiciel au début de son histoire peut être considéré comme une îe. Une île car les logiciels avaient peu de dépendances avec l'extérieur : pas de connexion en réseau, une interface simple clavier et souris… Cela n'offrait pas beaucoup de possibilité au logiciel mais rendait la problématique environnementale assez simple. Pourquoi ? Car compte tenu du nombre limité de dépendances le logiciel n'interférait pas avec d'autres logiciels ou matériels. Le seul impact était une consommation de ressources, qui était assez bien maîtrisée car l'unité logicielle était seule maîtresse de la consommation. Cette vue est assez simple mais ce que nous essayons de montrer, c'est qu'un logiciel n'ayant pas de dépendance avec l'extérieur a un impact limité (voir maîtrisé).

Ce type de logiciel « île » se retrouve en partie dans des logiciels embarqués assez simples : logiciel de gestion de lecteur MP3, logiciel de gestion de clé USB… Le logiciel est simple et les interfaces sont limitées : mémoire, interface USB…

L'archipel : l'ère des OS

L'ère moderne des systèmes d'exploitation est arrivée rapidement. Les systèmes d'exploitation facilitent l'utilisation du matériel, offrent une interface de programmation simplifiée… Cependant, contrairement à l'ère du logiciel « île », les dépendances sont beaucoup plus nombreuses : entre logiciels,

entre logiciels et drivers… La complexité pour le développeur du logiciel est de maîtriser cette multiplicité d'interfaces. Et cette problématique est très difficile à gérer : comment le concepteur du logiciel peut-il anticiper les logiciels, les drivers et autres éléments avec lesquels il communiquera ?

Le résultat de cette problématique peut se voir dans la perturbation de la gestion d'énergie par certains logiciels et drivers. Il est courant dans des systèmes d'exploitation que la gestion d'énergie ne fonctionne plus correctement. Par exemple, il est possible que la mise en veille ne s'active plus compte tenu d'un driver qui ne prend pas en compte les différents niveaux de gestion d'énergie du système ou les différents états du processeur. Autre exemple, pour un logiciel dont l'utilisateur a demandé la minimisation mais qui reste actif, la mise en veille ne se produira pas.

Avec les systèmes d'exploitation, le logiciel est donc passé d'une ère où il était isolé telle une île à une ère où il est confronté à une multitude d'interactions : l'ère des archipels. Cette ère est d'autant plus complexe pour le développeur qu'il faut gérer toutes les finesses des systèmes d'exploitation et toutes les possibles dépendances avec d'autres logiciels et interfaces. Sans cette gestion, il est certain que les impacts négatifs sur l'environnement sont nombreux : augmentation des ressources nécessaires, perturbation de la gestion d'énergie…

Le continent : l'ère de la convergence

Ces interdépendances ne s'arrêtent pas juste au niveau du système PC seul mais se prolonge hors du PC lorsqu'on le connecte avec le reste du monde. Et ceci s'est produit lorsqu'on a implémenté des possibilités aux systèmes de communiquer entre eux. Ethernet, wifi, Bluetooth… L'effet potentiellement négatif vu dans l'ère des archipels se retrouve ici dans l'ère des continents. Puisque le PC peut communiquer avec une multitude de matériel, il peut donc perturber leur fonctionnement.

Prenons l'exemple d'un réseau Ethernet où sont connectés de nombreux PC : un PC émet et demande des informations qui vont transiter sur le réseau. Ces informations vont avoir un impact sur l'environnement : réveil de certains matériels comme des serveurs, charge de traitement des autres PC, augmentation de la consommation des matériels réseaux, non mise en veille de certains matériels.

Le constat peut aller plus loin si on prend les moyens de communication sans fils. On doit prendre en plus de l'impact des informations, l'impact des ondes. Même si ce sujet est controversé (réel impact sur la santé, manque d'étude à long terme…), les ondes existent et ont un impact (même si très faible). Une mauvaise implémentation de la gestion de ces communications peut donc avoir un effet non maîtrisé par le développeur. L'effet d'échelle, compte tenu du nombre croissant de moyens de communication et d'appareils communicants, peut amener à une situation non tenable. Le logiciel a dans cette problématique une place clé.

L'océan : l'ère du cloud

L'ère actuelle est encore plus complexe en termes d'impact sur l'environnement. La mise en réseau de tous les PC comme le permet internet offre une multiplicité de possibilités, mais revers de la médaille, c'est que les services maintenant offerts et les transferts d'informations associés ont un impact important sur l'environnement. Pages internet de plus en plus lourdes, data centers consommant comme des usines… On appelle cette ère , « l'ère du cloud » (nuage) nous préfèrons l'appeler ère de l'océan. Nous allons essayer de découvrir les profondeurs (sombres ?) de cet océan.

Concrètement…

Un logiciel efficace dans la prise en compte du contexte est un logiciel qui s'adapte à son environnement. Sans cette adaptation, le logiciel répondra difficilement aux sollicitations

extérieures et ne se mettra pas dans un bon état de consommation. L'exemple concret est le changement de la luminosité de l'écran lors du branchement/débranchement de l'alimentation. Une application doit donc répondre aux changements de l'environnement et cela demande de prendre en compte le plus possible les informations venant de l'extérieur du logiciel :

- ✓ Données issues d'autres logiciels
- ✓ Capteurs et des actionneurs
- ✓ Données du système
- ✓ Hypothèses ou données sur l'utilisateur

Nous allons donc voir dans ce chapitre quelques bonnes pratiques concernant la prise en compte du contexte.

Prise en compte des transitions de mise en veille

Une des prise en compte de contexte la plus importante est celle de la gestion d'énergie. Pour atteindre les buts de la gestion d'énergie, les applications doivent être “Power-Aware” (conscient de la gestion d'énergie). Si cette gestion n'est pas optimisée, le logiciel perturbera la gestion de l'énergie du PC et la consommation d'énergie sera très importante. De plus, cela pourra créer un sentiment négatif de l'utilisateur envers le système global et envers la gestion de l'énergie.

Le programme doit donc être capable de :

- ✓ Supporter les événements de changement de mode d'énergie : par exemple prendre en compte le passage en mode veille prolongée
- ✓ Adapter le comportement à la configuration de la gestion

d'énergie et l'état courant du processeur : par exemple si le processeur diminue sa performance, le logiciel doit s'adapter.

✓ Intégrer la politique de gestion énergétique du système

Mais que se passe-t-il si on ne supporte pas les changements de mode ?

1. Le composant applicatif pose un veto sur la mise en veille

2. La veille ne se déclenche pas

3. Le système ne répond pas comme le demande l'utilisateur

4. L'utilisateur est frustré

5. Il désactive la gestion de l'énergie

Voici une liste des causes de blocage les plus fréquentes. Il faut particulièrement faire attention à ne pas intégrer de tels blocages dans son programme.

✓ Les drivers posent un veto à la mise en veille

✓ Les services ne se mettent pas en veille

✓ Les applications ne pouvant pas se mettre en veille ne créent pas de message

✓ Les applications émettent un message mais non visible (l'écran du PC est fermé par exemple)

Pourquoi les développeurs utilisent-ils leur droit de veto ?

✓ La prise en compte de la gestion de l'énergie demande un

développement supplémentaire et une expertise que le développeur ne possède pas

- ✓ Il faut tester cela dans les conditions cibles (différents modes de veille)
- ✓ Pour éviter de perdre la connexion réseau

L’OS est le maître de la gestion, pas l’application. Les applications doivent donc éviter de modifier les mécanismes de mise en mode basse consommation

- ✓ Evitez d’augmenter la fréquence des ticks d’horloge (ou alors la restaurer)
- ✓ Evitez d’utiliser des boucles d’attente (polling)
    - ✓ Utiliser plutôt les événements et registre
    - ✓ Les réduire en mode batterie
- ✓ Vérifier l’impact de l’application sur les états du processeur
- ✓ Evitez les accès réguliers au disque (Fonction Auto-save…)

Répondre et s’adapter aux événements de gestion d’énergie

En plus des états de veille, les événements de gestion d’énergie telles que les alertes issues des états batterie, sont à prendre en compte. Il faut interagir avec la politique de gestion de l’énergie pour pourvoir s’adapter. Les applications ont donc besoin d’avoir accès aux événements de la gestion d’alimentation :

- ✓ Changement AC/DC

- ✓ Etat de la batterie
- ✓ Moniteur On/Off

Il faut dans ce cas utiliser les événements mis à disposition de l'OS et ne pas faire du polling. L'exemple de l'affichage est dans ce cadre, un exemple typique de contrainte pour une application. Un programme est généralement fait pour être affiché, il faut donc faire surveiller quand l'utilisateur en a besoin ou pas. Les événements de la gestion d'énergie seront donc de bonnes aides pour pouvoir s'adapter. Ensuite :

- ✓ Si l'application est minimisée, il ne faut pas bloquer la mise en veille de l'affichage
- ✓ Pas la peine d'activer les moteurs graphiques si l'écran est éteint

Adapter le comportement à l'état du système

Au-delà de la gestion de l'énergie, le logiciel doit continuellement s'adapter à l'état du système et à l'ensemble des autres logiciels. Les informations de contexte sont généralement liées à l'environnement dans lequel évolue le logiciel. Si le logiciel est une application web 2.0, les informations du contexte seront l'état de la connexion, des informations sur le navigateur…

OUVERTURE DES DONNEES

Format de fichier ouvert ou propriétaire

Problème de la fermeture des formats

Le logiciel peut respecter les principes du libre (Voir chapitre « Libre, gratuit, open source » p.118) mais ce n'est pas le choix de tous les développeurs. Cependant sans ouvrir le code ou sans suivre ce mouvement, l'ouverture de certaines données est possible. Par exemple, l'ouverture du format des fichiers générés par le logiciel est une voie à explorer. Deux choix sont possibles pour les concepteurs concernant les fichiers de sortie : soit utiliser un format fermé soit un format ouvert. Les formats fermés sont généralement appelés propriétaires car ils sont conçus par une société unique et les fichiers générés contiennent des données qui ne sont pas lisibles sans un logiciel fournit par la société. Comme vu pour les langages propriétaires (Voir chapitre « Langages » p.155), cela pose la question de la durabilité des fichiers et des logiciels. Si la société détentrice du format décide de ne plus maintenir les logiciels permettant de générer et lire le fichier, alors on peut s'attendre à ce que les fichiers ne soient plus utilisables et lisibles à moyen terme. Le problème est le même que pour les supports physiques : que faire d'une disquette si l'on n'a plus de lecteur ? De plus, la multiplication des formats amène à un besoin croissant en fonctionnalité et en programme : Afin de pouvoir lire tous les fichiers possibles, il est nécessaire d'installer soit des extensions, soit des nouveaux logiciels. L'utilisateur est ainsi bloqué dans le choix de ses outils par le format du fichier.

Formats ouverts

Il est préférable d'utiliser des fichiers aux formats ouverts. Ces formats sont indépendants d'un logiciel ou d'une société et les

données sont lisibles et non encodées. L'utilisation de tels formats améliore l'interopérabilité pour les échanges de fichiers entre les utilisateurs. On peut citer comme formats ouverts par exemple :

- ✓ RTF et ODT pour le texte
- ✓ SVG pour les images
- ✓ OGG pour la vidéo

Attention comme pour le logiciel libre, il existe différentes variantes qui ne respectent pas totalement l'ouverture : des logiciels utilisant des formats ouverts mais ajoutant des extensions propriétaires, des formats ouverts mais propriétaires (par exemple Xvid pour les vidéos).

Lors du développement de logiciel, on devra donc de préférence utiliser comme format principal un format ouvert existant. Si aucun format ouvert n'existe (Cependant peu probable), la modification d'un format existant ou la création d'un nouveau format ouvert sera possible. Dans le cas du choix d'un format propriétaire, la prise en charge des formats ouverts sera nécessaire pour offrir à l'utilisateur un choix alternatif.

Données ouvertes (Open Data)

Définitions des données ouvertes

Le mouvement « Open Data » vise à rendre les données publiques accessibles à tous. Il préconise une libre disponibilité pour tous des données sans restriction d'utilisation ou d'autre contrôle. Une donnée publique peut être considérée comme[47] :

- ✓ Une donnée collectée par un organisme public ou une

[47] Libertic : http://www.slideshare.net/libertic/lopendata-5128072

entreprise

- ✓ Une donnée non-nominative
- ✓ Une donnée ne relevant pas de la vie privée ou de la sécurité

Les intérêts de ce mouvement sont multiples : Transparence des données publiques, meilleure collaboration et intégration des populations dans la vie publique, efficacité et amélioration des services publiques… Tout système informatique traitant des données publiques a donc intérêt à publier ses données comme des données ouvertes. Cela permet d'améliorer le partage des informations publiques vers le public mais aussi avec d'autres données ouvertes et ainsi multiplier les modalités d'utilisation de ces données. L'intérêt n'est pas que pour l'utilisateur mais pour le système lui-même : L'ouverture des données va amener une consolidation et une amélioration des données et donc une meilleure efficience du système.

Principes des données ouvertes

L'ouverture des données nécessite de respecter certains principes. Sans ces principes, on pourrait estimer que toutes les données qui sont sur internet sont ouvertes. Le groupe de travail Open Governement Data a défini 8 règles pour évaluer l'ouverture des données. Les données ouvertes doivent être :

- ✓ Complètes
- ✓ Primaires (c'est-à-dire telles que collectées à la source)
- ✓ Opportunes
- ✓ Accessibles
- ✓ Exploitables

- ✓ Non discriminatoires
- ✓ Non propriétaires
- ✓ Libres de droits

Tim Berners Lee, fondateur du W3C, propose une méthodologie pour l'évaluation des données ouvertes, basée sur 5 étoiles[48] :

- ✓ 1 étoile : Données accessibles sur le web (sans conditions de formats)
- ✓ 2 étoiles : Données accessibles structurées (exemple: Excel au lieu de l'image d'un tableau)
- ✓ 3 étoiles : Formats non-propriétaires (exemple: csv au lieu d'Excel)
- ✓ 4 étoiles : Usage d'URL pour identifier les données
- ✓ 5 étoiles : Données liées sémantiquement

Intégration dans les systèmes et logiciels

L'open data est une pratique qui permet de répondre à des aspects importants du développement durable (pilier social en particulier mais aussi l'aspect environnemental avec par exemple l'ouverture des données de transport). Quand les systèmes informatiques traitent des données publiques, il est intéressant les ouvrir. Ceci peut se faire une fois le système développé, mais il sera moins coûteux de le faire dès la conception.

[48] http://inkdroid.org/journal/2010/06/04/the-5-stars-of-open-linked-data/

Pour aller plus loin

Association Libertic : http://libertic.wordpress.com/

LIEN ENTRE LES PATTERNS

On associe souvent gain de performance et efficience énergétique. Ceci n'est pas une règle générique. En effet, il est certain que si un programme est performant, il sera exécuté plus rapidement. Cependant, si le programme utilise des mécanismes qui sont très consommateur pour être performant, on ne peut pas dire qu'il sera efficient énergétiquement. On peut prendre l'exemple d'un programme qui fera fonctionner le processeur dans l'état de consommation la plus importante pour réaliser une tâche très rapidement. Ce programme pourrait utiliser un état de consommation plus faible du processeur. La tâche sera alors réalisée moins rapidement mais utilisera moins d'énergie. L'avantage énergétique entre les deux solutions dépendra de nombreux paramètres : Durée de la tâche, niveau de consommation du processeur…

Plus globalement, l'efficience énergétique est une caractéristique parmi d'autres. Le développeur doit doser toutes ses caractéristiques pour répondre aux besoins des utilisateurs. Parmi ses caractéristiques, on peut lister : Sécurité, performance, maintenabilité… Voir chapitre « Qualité logicielle et ISO 25000 » p.95 pour la classification ISO 25000 des caractéristiques. Or ces caractéristiques ne sont pas toujours en accord. Rendre une application sure (au sens sécurité) demandera de mettre des mécanismes dans le code qui seront consommateurs en ressource et qui iront dans le sens d'une efficience énergétique faible.

Idées d'action Green Code Lab

Expérimenter l'effet des différentes caractéristiques sur l'efficience énergétique.

Le dosage entres caractéristiques doit se faire en accord avec les attentes des utilisateurs. Il est inutile de rendre le logiciel trop performant si l'utilisateur n'en a pas besoin. Le problème est que le développeur tente de couvrir toutes les caractéristiques en même temps. Cela a un impact du point de vue budgétaire mais surtout du point de vue énergétique. Voir chapitre « Expression de besoin » p. 108 pour l'identification des besoins.

LE GREEN IT S'APPLIQUE AUSSI AUX FABRICANTS DE LOGICIELS

En appliquant les bonnes pratiques du développement durable et du Green IT, éditeurs et SSII peuvent diviser par deux l'empreinte carbone de leur logiciel[49].

On considère trop souvent, à tort, qu'un logiciel est immatériel. Sa fabrication nécessite pourtant de nombreuses ressources physiques : bâtiment hébergeant les développeurs, kilomètres parcourus entre le domicile et le travail, matériel informatique, électricité, climatisation, etc. Chez la plupart des « fabricants » de logiciel - SSII et éditeurs - l'empreinte écologique se concentre par ordre d'importance dans la construction des bâtiments (hors bâtiments basse consommation), les déplacements des collaborateurs, et la fabrication du matériel informatique.

En plus de pratiquer l'éco-conception des logiciels, SSII et éditeurs doivent donc adopter les gestes essentiels du Green IT, et plus largement du développement durable (éviter les déplacements inutiles en collaborant à distance, réduire l'empreinte des bureaux, etc.). Dans le domaine du Green IT, toutes les analyses de cycle de vie démontrent que l'empreinte écologique et sociale se concentre dans la fabrication et la fin de vie des équipements. Ce constat est identique quel que soit l'équipement : ordinateur de bureau, serveur, téléphone, etc.

Les gestes clés du Green IT consistent, par ordre d'importance décroissante, à :

- ✓ utiliser le plus longtemps possible son matériel informatique (ordinateur, écran, serveur, etc.),

[49] GreenIT.fr, Frédéric Bordage, 2011

en fin de vie, gérer sa collecte et son retraitement auprès de professionnels respectueux de l'environnement et qui privilégient le reconditionnement,

✓ acheter du matériel d'occasion reconditionné ou, à défaut, éco-conçu (écolabel EPEAT Gold),

✓ réduire la consommation électrique.

En appliquant les gestes clés du développement durable et du Green IT, SSII et éditeurs peuvent réduire considérablement l'empreinte écologique de leurs logiciels.

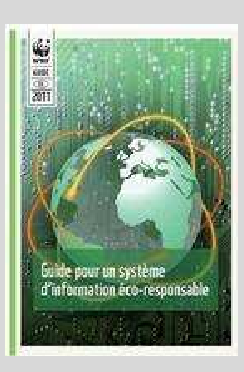
Pour ne prendre qu'un exemple, l'empreinte carbone d'un logiciel est constituée de l'ensemble des émissions de gaz à effet de serre (GES) liées à la construction des bâtiments, aux déplacements des développeurs, à la fabrication du matériel informatique, et à la production d'électricité consommée par les bâtiments et les équipements informatiques. En évitant les déplacements inutiles et en allongeant la durée de vie des équipements informatiques, il est possible de réduire de façon importante l'empreinte carbone de la conception d'un logiciel.

Par ailleurs, en utilisant un matériel informatique un peu ancien pour développer et tester le logiciel, l'éditeur ou la SSII sont contraints de s'adapter aux limites physiques des équipements de leurs clients. Rien de tel pour être sensibilisés et ainsi réduire l'empreinte ressources de leurs développements.

Pour aller plus loin

GreenIT.fr : site de référence sur la green IT et les TIC durables, GreenIT.fr propose une liste de bonnes pratiques adaptées à la gestion des systèmes d'information.

WWF : Guide pour un système d'information éco-responsable édité par le WWF et réalisé en partenariat avec GreenIT.fr et le groupe Eco-Info du CNRS.



5 Après le développement

On résume trop souvent une meilleure conception du logiciel en parlant de la phase d'utilisation mais l'analyse du cycle de vie d'un logiciel montre que ce n'est pas tout. Il est important d'appliquer des bonnes pratiques dans toutes les phases de vie du logiciel : validation, fin de vie… Sans quoi les efforts mis en œuvre lors du développement pourraient être annulés rapidement.

VALIDATION

Qualité logicielle

Introduction

La qualité logicielle est le niveau de confiance que l'on a dans le logiciel. Ce niveau de confiance peut être évalué avec plusieurs techniques et plusieurs indicateurs. Nous avons vu dans le chapitre « Qualité logicielle et ISO 25000 » p.95, la normalisation de ce domaine et les axes d'évaluation associés. Améliorer la qualité logicielle va permettre de s'assurer que le logiciel répondra aux besoins des utilisateurs et que son comportement sera maîtrisé dans toutes les conditions d'utilisation. La qualité va être mesurée et maîtrisée en appliquant des techniques de validation et de vérification du logiciel. La validation va permettre de s'assurer que le logiciel respecte bien le fonctionnement attendu et la vérification si le logiciel fonctionne correctement. Voici quelques exemples de techniques :

- ✓ Relecture du code
- ✓ Analyse statique du code
- ✓ Tests unitaires
- ✓ Validation fonctionnelle
- ✓ Tests de performance
- ✓ Recette du produit

De point de vue du développement durable, ces techniques vont permettre d'atteindre plusieurs objectifs :

- ✓ Maîtrise de l'impact du processus de développement
- ✓ Vérification de l'implémentation des green patterns
- ✓ Amélioration de la qualité du logiciel et donc de son efficacité

Maîtrise de l'impact du processus de développement

Le processus de développement du logiciel doit être maîtrisé pour que la phase de fabrication ait le moins d'effet possible sur l'environnement. Or nous avons précisé dans le chapitre « Inefficacité du processus logiciel » p.50 que ce processus était mal maîtrisé. La validation et la vérification vont permettre de mesurer la qualité et l'atteinte des objectifs au fur et à mesure de l'évolution du développement. Cette validation devra être appliquée dès le début du projet pour qu'elle soit la plus efficace. En effet, valider le code une fois le développement terminé ne permettra pas de gérer le processus de développement et amènera des corrections trop coûteuses. En effet, plus une anomalie sera détectée tard dans le processus, plus l'effort de prise en compte sera important (Compte tenu de l'analyse, de la correction, de la vérification). Le coût entre une détection d'anomalie en phase de développement et en phase d'utilisation peut aller de 1 à 100. Le coût environnemental suit alors le même phénomène : utilisation des ressources pour analyser et corriger les bugs, déplacements des experts, impacts sur d'autres projets. Imaginez l'impact d'un bug détecté sur le calculateur d'une voiture alors qu'elle est déjà commercialisée : Les développeurs vont devoir appliquer un processus de livraison du logiciel vers tous les garages et des milliers d'utilisateurs vont devoir ramener leur voiture au garage et vont donc émettre du CO^2…

Nous n'entrerons pas dans les détails des techniques de validation dans ce manuel cependant toute méthode ou processus qui intègre les tests au plus tôt sera bénéfique pour

la qualité. L'attention se fera même plus en amont jusqu'à l'identification des exigences et à l'expression de besoin.

Vérification de l'implémentation des green patterns

La validation et la vérification vont permettre de contrôler que l'implémentation des green patterns agissent comme attendu. L'intégration des bonnes pratiques n'ont pas forcément d'effet directement visible par le développeur. Il est donc nécessaire de mettre en place un plan de validation qui vérifiera l'atteinte des objectifs. On pourra vérifier par exemple :

- ✓ L'amélioration de l'efficience énergétique
- ✓ La diminution du temps d'exécution d'une tâche
- ✓ La consommation de mémoire
- ✓ L'accessibilité de l'interface

Vous trouverez dans ce chapitre ces validations et outils spécifiques aux green patterns.

Amélioration de la qualité du logiciel et donc de son efficience

Plus généralement, la validation des applications améliore la qualité du logiciel est permettent de répondre directement aux exigences de l'éco-conception. Une meilleure qualité amènera par exemple une meilleure satisfaction pour l'utilisateur et donc une durabilité pour l'application. De nombreuses pratiques de validation vérifieront directement l'efficience de l'application :

- ✓ Tests de performance des applications web qui vérifieront que le site répond correctement avec un nombre d'utilisateur important

- ✓ Audits des sites par rapport aux normes d’accessibilité W3C.

Mesure de la consommation

Introduction

« Ce qui ne peut pas être mesuré, ne peut être géré » (Lord Kelvin). Voici une célèbre expression qui s’applique à notre domaine. Si l’on ne connaît pas réellement l’impact du logiciel sur l’environnement alors nous ne pouvons pas savoir si les green design patterns que nous codons sont bénéfiques. Il est probable que sans mesure les green patterns codés auront le même effets qu’une méthodologie « Quick and Dirty » (Voir chapitre « Quick And Dirty » p.86). La mesure la plus courante est la mesure de la consommation d’énergie mais d’autres mesures peuvent être appliquées. L’équivalent émissions CO^2 par exemple aura le bénéfice de pouvoir intégrer des émissions autres que la consommation du système informatique : transport des supports physiques, déplacement des personnes… A la notion de consommation (ou d’émission) pourra être ajoutée celle de performance. L’efficacité des pratiques pourront alors être évaluée : watt par instructions ou par requêtes…

Cette mesure n’est pas simple. Premièrement, la consommation peut être mesurée au niveau du système. La mesure que l’on a est alors une mesure de la consommation totale. Le problème est que l’on veut évaluer la consommation au niveau d’un logiciel, et pas au niveau du système global (système d’exploitation et tous les logiciels). Quelques solutions existent mais ne sont pas généralisées. Ceci s’explique par le fait que la mesure de la consommation n’est pas une chose commune dans le développement logiciel. Il est donc normal que l’offre en outils et méthodologies ne soit pas répandue.

Données de mesure physiques

Il est important pour le développeur de connaître les unités physiques de mesure. En effet, ses métriques habituelles de travail sont plutôt les octets et les lignes de code. Pour la mesure de consommation d'énergie, il va falloir mettre en jeu d'autres données. Voici un les métriques à prendre en compte :

Ampère

Il s'agit de l'unité de mesure de l'intensité de courant mesurée en charge électrique par seconde. Cette unité n'est pas utilisée pour mesurer la consommation mais plutôt dimensionner les composant.

Watt

Le Watt est l'unité mesurant la puissance, c'est-à-dire le taux d'énergie consommée par l'appareil. La puissance est obtenue en multipliant la tension par l'intensité pour le courant continu. Pour le courant alternatif, la formule est un peu plus complexe car elle fait intervenir le déphasage du courant. La puissance est une mesure instantanée, c'est à dire qu'elle donne le taux d'énergie consommé à un instant donné.

Wattheure

Le Wattheure mesure une quantité d’énergie. Un Wh est la quantité d’énergie consommée par un équipement d’une puissance d’un Watt pendant une heure. Un appareil d’une puissance de 100W, consommera 50 Wh d’énergie pendant 30 mn ou 200 Wh pendant 2 heures. Contrairement à la puissance, l'énergie consommée fait toujours intervenir une période de temps définie. On estime par exemple la consommation électrique d’un poste de travail en nombre de kWh par an. La puissance et l'énergie sont souvent confondues. Comparer deux puissances ne permet pas de donner une conclusion sur l'efficience énergétique car il faut

faire intervenir l'utilisation dans le temps.

Outils de mesure physique

La mesure des données de consommation nécessite des instruments spécifiques. L'estimation est possible à partir de données constructeurs mais est compliquée car les données disponibles ne sont pas assez précises. En effet, pour évaluer la consommation d'un logiciel, il faut pouvoir associer à une instruction et à l'état du processeur la consommation. Or les données fournies ne donnent que les consommations du système dans quelques états.

La mesure sera d'autant plus précise que plusieurs points de mesure seront faits. Idéalement, il faudrait effectuer des mesures indépendantes du processeur, de la mémoire, des disques durs et des autres périphériques. Cette pratique n'est pas simple car elle nécessite de mesurer le courant à l'intérieur du système et les alimentations des éléments ne sont généralement pas accessibles. Pour la mesure sur des systèmes complexes (avec plusieurs PC et périphériques autonomes), la pratique est plus facile. On mesurera par exemple la consommation du PC mais aussi du routeur Ethernet associé pour connaître l'impact sur la consommation du réseau.

Multimètres

Les appareils mesurant la tension et l'intensité sont utilisables et permettent d'avoir des données qui serviront à calculer la puissance. On peut citer les multimètres, les analyseurs logiques, les voltmètres... Le problème de ces instruments est qu'ils ne donnent pas directement la puissance et l'énergie. Des post-traitements des données récupérées permettront d'obtenir les résultats.

Wattmètres domestiques

Les wattmètres sont des appareils qui calculent directement la consommation et la puissance. Ils se branchent entre la prise de courant et l'appareil. Ses appareils sont peu coûteux (environ 30 €). On peut citer par exemple Brennenstuhl et Watt's up.



Figure 33 - Wattmètre

Idées d'action Green Code Lab

Les appareils de mesure ont généralement des API pour les commandés. L'intégration de ces API dans des SDK comme Intel Energy Checker est très utiles pour effectuée des mesures. La mise en place d'une telle intégration est à faire.

Wattmètres évolués

Les wattmètres domestiques ne permettent pas d'enregistrer les données. Les wattmètres professionnels permettent soit de récupérer les données par carte mémoire ou d'être relié à un PC. Ces fonctionnalités sont importantes si on veut analyser précisément la consommation et effectuer des corrélations avec d'autres données (Charge CPU...). Voici une liste de tels matériels :

- ✓ Volcraft Energylogger 4000 – récupération par carte mémoire SD
- ✓ Watt's up meter Pro – Liaison temps réel avec PC
- ✓ Plugwyse – Liaison radio Zeegbee



Figure 34 - Watt'up Meter

Analyseurs logiques

Les appareils de mesure de laboratoire comme les analyseurs logiques sont plus précis et plus évolué mais plus coûteux. Ils pourront s'intégrer dans un banc de tests logiciels.

Outils de mesure logiciels de la consommation

Idées d'action Green Code Lab :

Intel Energy Checker n'est pas simple, un petit tutoriel avec un projet exemple serait nécessaire

Intel Energy Checker peut se coupler à des outils de mesure. L'intégration d'outils plus rependus serait bienvenue (Plugwyse par exemple !)

Green Fox nécessite des améliorations.

Il est probable que d'autres logiciels existent pour d'autres langages ou technologies. Le page ressource du green code lab est la pour ça. Et si un nouveau framework est nécessaire. Pourquoi ne pas le faire dans le code lab en licence

La mesure de la consommation d'une partie du logiciel nécessite d'être intrusif. Le logiciel de mesure doit en effet mettre des sondes pour mesurer les ressources utilisées. Ceci n'est rien d'autre qu'un classique profiling. Le profiling est utilisé pour analyser le temps passé par différentes parties du logiciel. On peut citer : Netbean profiler, X Debug… Un logiciel de mesure (ou plus généralement la méthodologie associée) doit effectuer la même analyse. La problématique pour lui est d'évaluer la consommation à partir du profiling réalisé. Entre alors en jeu des modèles de consommation. Ces modèles caractérisent la consommation d'énergie en fonction de mesure : charges CPU, nombres d'instructions… On obtient alors la consommation d'énergie d'une tâche du programme. Tel est le fonctionnement des logiciels de mesure existant :

- ✓ Intel Energy Checker[50] : C'est un SDK qui est fourni par INTEL est qui permet au développeur d'instrumenter son logiciel. Le SDK se connecte à certains instruments de mesure physique (Watt's Up Meter par exemple) et peut intégrer d'autres matériels.

- ✓ Powertutor : C'est un logiciel développé par l'université du Michigan[51] qui permet de fournir la consommation globale du mobile et des applications pour les téléphones Android. Des fichiers de log peuvent être enregistrés et rapatriés sur PC.

[50] http://software.intel.com/en-us/articles/intel-energy-checker-sdk/
[51] http://ziyang.eecs.umich.edu/projects/powertutor/

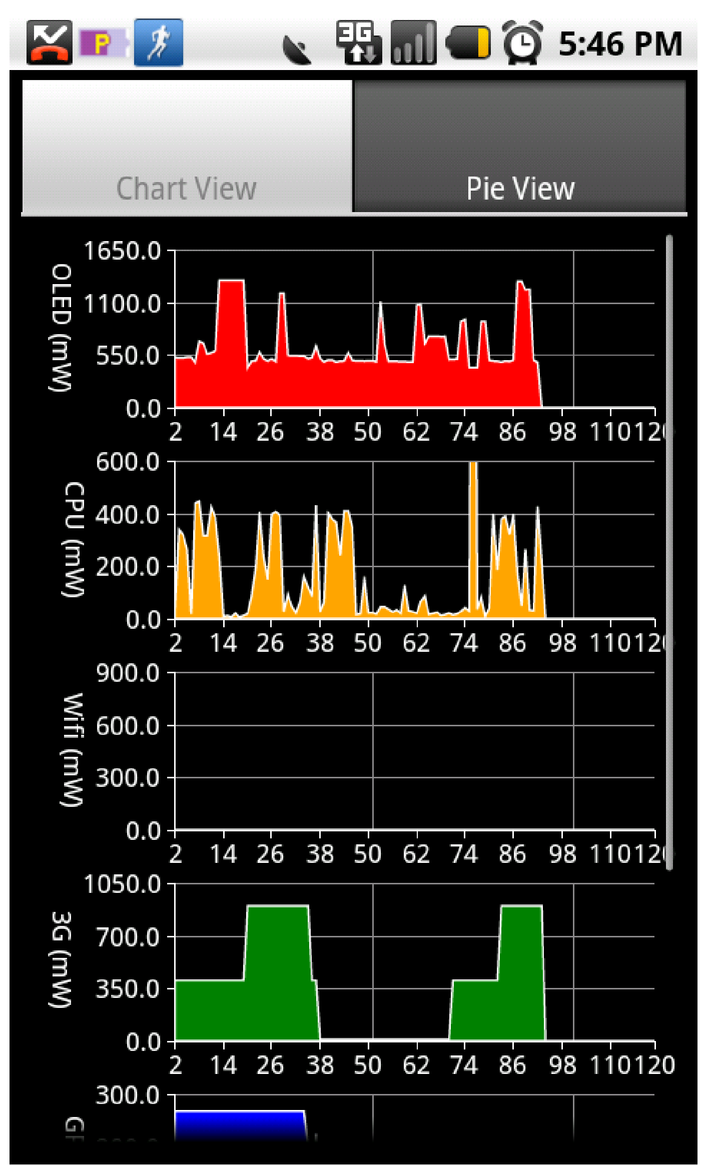


Figure 35 - PowerTutor

- ✓ Green Fox : Module pour Firefox permettant de mesurer la consommation d'une page ou d'une application web. Ce module a été développé pour le challenge USI.

Pour d'autres logiciels, allez voir dans les parties spécifiques technologies de ce livre. Il existe en effet de nombreux logiciels qui sont spécifiques aux technologies et aux appareils.

Puisque ces mesures sont très dépendantes de la plate-forme matérielle, il faudra faire particulièrement attention aux conclusions et à la généralisation. En effet, un green pattern pourra avoir un bénéfice important sur une plate-forme A mais être minimisé sur une plate-forme B. D'où la nécessité de classifier les green patterns en fonction de la dépendance à la plate-forme matérielle.

Une méthode plus sommaire est bien sûr la mesure globale du comportement du système. Cette méthode conviendra par exemple pour comparer deux logiciels ou alors pour stresser le logiciel dans le contexte global du système. On pourra donc difficilement voir l'effet d'un green pattern mais l'effet de tous les green patterns dans le système final. Cette méthodologie peut être appliquée en effectuant une mesure avec le système vierge (i.e. OS et autres applications installés au repos) puis avec le logiciel installé. La différence permet d'avoir la part de consommation au repos du logiciel et la part en activité.

Outils de mesure logiciels indirects

Les processeurs offrent des informations sur l'état du processeur comme l'état de veille (C1 state...) et sur d'autres informations importantes. Ces informations permettent d'analyser l'efficience du logiciel. C'est une solution plus souple que les solutions de mesure avec wattmètre mais qui ne permettent pas de conclure sur la consommation des autres périphériques que la CPU.

- ✓ Ptop : Outil GNU/Linux qui évalue la consommation à partir

d'un modèle d'énergie prédéfini. L'outil permet de connaître la consommation d'un processus unitaire. Les résultats obtenus sont très proche d'une mesure

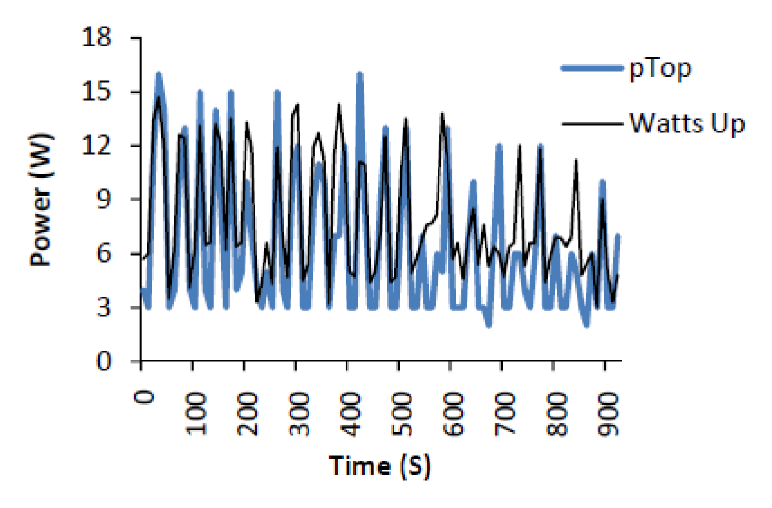


Figure 36 - Comparaison pTop et mesure physique

Pour aller plus loin

Explication de l'outil et de son utilisation par l'équipe ayant développé l'outil http://www.sigops.org/sosp/sosp09/papers/hotpower_13_do .pdf

✓ Intel PowerInformer : Ce SDK ne fournit pas d'information sur la consommation mais sur des données impactant la consommation (Attention cette application ne semble pas être maintenu par Intel).

- Etat du processeur

- Percentage time of C1, C2 and C3 states of the system and of all logical processors
- Average residency of C1, C2 and C3 states of the system and of all logical processors
- Percentage time of the system and of all logical processors
- Percentage time of the P stages of the system
- System calls per second
- Interrupt rate of the system
- Disk I/O operation (access, read and write) rates
- File I/O operation (control, read and write) rates

✓ Microsoft Event Tracing for Windows (ETW) : Solution de tracing pour les application C ou C++ sous Microsoft Windows qui permet d’obtenir des informations sur les événements d'une tâche.

✓ PowerTop : analyse des temps passés par les applications GNU/LINUX dans les différents états du processeurs.

```
     PowerTOP version 1.13      (C) 2007 Intel Corporation

Cn                  Avg residency       P-states (frequencies)
C0 (cpu running)        ( 0.0%)         Turbo Mode     0.0%
polling             0.0ms ( 0.0%)         2.81 Ghz     1.2%
C1 mwait            0.0ms ( 0.0%)         2.14 Ghz     0.0%
C2 mwait            0.1ms ( 0.0%)         1.60 Ghz     0.0%
C4 mwait           62.9ms (100.0%)         800 Mhz    98.7%

Wakeups-from-idle per second : 17.4     interval: 20.0s
no ACPI power usage estimate available

Top causes for wakeups:
  41.7% (  9.2)   [kernel core] hrtimer_start (tick_sched_timer)
  17.3% (  3.8)   [extra timer interrupt]
  12.1% (  2.6)   [eth0] <interrupt>
  11.6% (  2.5)   [kernel scheduler] Load balancing tick
   5.2% (  1.1)   [acpi] <interrupt>
   4.6% (  1.0)   events/0
   2.3% (  0.5)   events/1
   0.9% (  0.2)   [kernel core] dev_watchdog (dev_watchdog)
   0.9% (  0.2)   init
   0.7% (  0.1)   upowerd
   0.5% (  0.1)   flush-btrfs-2
   0.5% (  0.1)   flush-btrfs-1
   0.5% (  0.1)   bdi-default
   0.5% (  0.1)   btrfs-transacti
   0.2% (  0.1)   btrfs-submit-0
   0.2% (  0.1)   syslogd

A SATA device is active 33.3% of the time:
host1

 Q - Quit   R - Refresh   S - SATA Link Power Management
```

Figure 37 - PowerTop pour GNU/Linux

Pour aller plus loin

Fine-Grained Application Analysis for Energy-Aware Computing – Intel – Utilisation de nombreux outils de mesure logiciel.

La charge CPU est une mesure indirecte qui doit être utilisée avec prudence. En effet, comme vu dans le chapitre « Efficacité de calcul » p.168, la consommation de deux tâches utilisant la

même charge CPU variera avec leur fréquence et leur durée. Il n'y a donc pas une relation simple entre une consommation et charge CPU même si dans la plupart des cas on peut avancer une corrélation forte.

Pour aller plus loin

Etude charge CPU et consommation (entre autre) - Energy-Efficient Platforms – Considerations for Application Software & Services.- http://software.intel.com/file/38273

Notion de performance

L'unique mesure de la consommation d'énergie ne suffit pas pour caractériser l'impact d'un logiciel. En effet, deux logiciels ayant les mêmes fonctionnalités pourront consommer la même énergie mais l'un pourra être plus efficace que l'autre (car il réalisera par exemple la tâche plus rapidement). Rapidité et consommation ne sont pas forcément tout le temps liées. Il est donc nécessaire de rapporter les mesures de consommation à ces mesures de performance. Ces mesures sont d'autant plus faciles à identifier qu'elles sont généralement répandues dans les projets logiciels (Temps d'exécution, nombre de requêtes). Ces notions de performance dépendront bien-sûr du contexte.

Évaluation de l'impact global

Les choix technologiques et green patterns doivent se faire en prenant en compte tous le cycle de vie du logiciel. Une optimisation même petite de la consommation d'un logiciel qui sera déployé sur la quasi-totalité des systèmes d'exploitation sera par exemple non négligeable par rapport à la même optimisation uniquement utilisée par un utilisateur. Au même titre, il ne serait pas nécessaire d'appliquer des green patterns qui rendraient la phase de fabrication plus polluante que la phase d'utilisation.

Concernant la phase de fabrication, toutes les ressources nécessaires au développement du logiciel sont à prendre en compte : ressources énergétiques des infrastructures, déplacement des équipes… Il ne faut surtout pas négliger des coûts qui sont cachés : équipes de support, projets pilotes… Ces derniers sont souvent oubliés alors qu'ils participent au développement du logiciel à proprement parler. De même, les projets logiciels utilisent de plus en plus de librairies et de framework. Par définition ses modules sont déjà développés. La prise en compte de ces éléments dans l'évaluation de l'impact de la phase de fabrication est donc à faire. Au final cette évaluation peut paraître complexe, mais le logiciel est un système complexe. Au même titre que le calcul de l'impact de la fabrication d'un PC, le calcul doit prendre en compte un processus de fabrication réparti entre plusieurs intervenants et avec des niveaux de sous-traitances importants.

Le deuxième phase la plus complexe à évaluer est celle de la phase d'utilisation. La mesure unitaire de la consommation du logiciel ne suffit pas. En effet, contrairement à un produit pour lequel on arrive à associer un ratio par rapport à la fabrication (1 pour 1), la fabrication est mutualisée mais la phase d'utilisation dépend d'un paramètre important : le déploiement. Prenons le cas d'un logiciel qui aurait un impact de 100 en fabrication et 1 en utilisation unitaire (Sur une phase de vie prédéfinie). Si le logiciel est déployé sur 10 postes, on aura 100 en fabrication et 10 en utilisation, soit 110 au total. Dans le cas d'un déploiement sur 1000 postes, on aura 100 en fabrication et 1000 en utilisation. Et comme le déploiement n'est pas maîtrisé, il est difficile de donner l'impact total en phase d'utilisation. Cela demande donc de réévaluer l'impact global du logiciel régulièrement. Une stratégie d'éco-conception initialement prévue faible parce qu'un déploiement à petite échelle était prévu devra être réévaluée (en implémentant de meilleures pratiques par exemple) dans le cas d'un succès du logiciel. Moralité, il est donc préférable d'être optimiste et de prévoir le succès du logiciel dès le début du projet (Voir chapitre « Quick And Dirty » p. 86) ! On évitera la malheureuse aventure de devoir changer par exemple un langage en cours de projet.

Vérifier la prise en compte du contexte, l'efficacité de calcul et de données

Il existe une grande quantité d'outil de mesure et de profiling qui permettent de vérifier les axes d'efficacité. Voici quelques exemples.

Pour aider le développeur, des environnements de développement permettent de facilement récupérer ces contextes et ensuite adapter le fonctionnement. On peut citer à titre d'exemple les outils suivant :

- ✓ Mobile Platform Software Developer Kit (MPSDK)
- ✓ Laptop Gaming Technology Developer Kit (Gaming TDK)
- ✓ Web 2.0 Technology Development Kit (Web 2.0 TDK)

Pour la mesure de la performance de la mémoire, les logiciels vont dépendre des technologies. On peut lister :

- ✓ Outil de profiling Chrome.[52]
- ✓ .Net memory profiler[53]
- ✓ Eclipe memory Analyser (MAT)[54]
- ✓ Yourkit pour java et .Net[55]

---

[52] http://code.google.com/intl/fr-FR/chrome/devtools/docs/heap-profiling.html
[53] http://code.google.com/intl/fr-FR/chrome/devtools/docs/heap-profiling.html
[54] http://eclipse.org/mat/
[55] http://www.yourkit.com/

DIFFUSION

Canaux de diffusion

La diffusion du logiciel n'est pas à négliger. La diffusion est en effet un aspect stratégique pour les éditeurs de logiciel. En effet une multiplicité de canaux de distribution et synonyme de visibilité. De plus les éditeurs offrent différents canaux pour pouvoir couvrir tous les besoins : supports optiques, clés USB, fichiers en ligne… Les supports optiques sont polluants. D'après greenit.fr : Un CD de 23 grammes est composé à 90% de polycarbonates : un thermoplastique noble, très résistant et surtout recyclable. Quant au boîtier, il est totalement recyclable. La couche de polycarbonate est transformée en granules qui deviendront des cartes plastiques, des bijoux, des lampes… Broyés, les boîtiers sont transformés en granulés de polystyrène puis convertis en nouveaux boitiers ou en badges. Quant aux fichiers en ligne, leur empreinte environnementale consistera essentiellement en espace disque occupé et en consommation induite sur le réseau.

L'analyse des canaux de diffusion doit donc prendre en compte les nombreux éléments qui permettent de distribuer le logiciel :

✓ Fabrication / diffusion par support optique

✓ Espace de stockage pour diffusion

✓ Manuel papier / électronique

✓ Charge réseau lors de la diffusion

Cet impact est d'autant plus importante que le mode de distribution des logiciels se fait en mode « retail » (support optique mis en magasin). Le mode de diffusion doit donc être analysée et maîtrisée. Une étude Intel / Lawrence Berkeley

montre très bien les impacts en fonction du mode de diffusion[56]. Elle différentie les modes suivants :

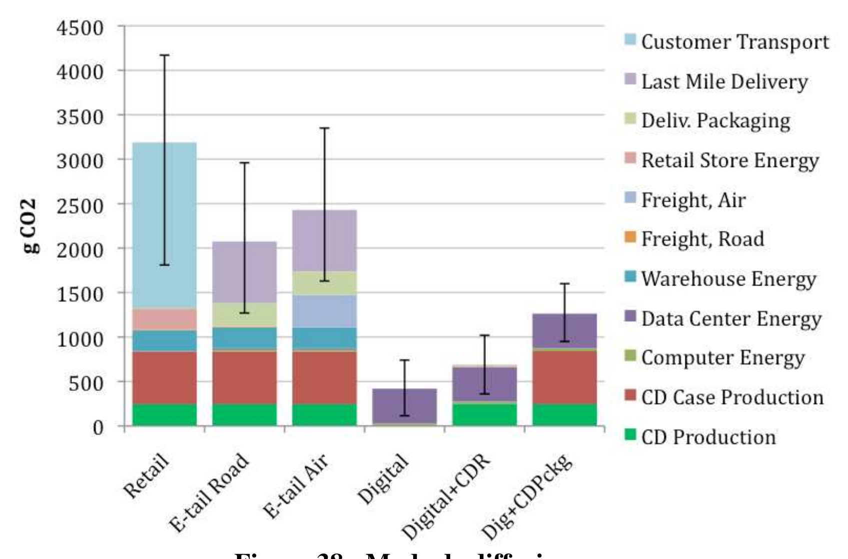


Figure 38 - Mode de diffusion

Cependant le mode numérique de diffusion doit bien être maîtrisé : impact des data centers où sont stockés les fichiers, impact du téléchargement. De plus, la complexité d'installation et l'impact de l'installation est à prendre en compte. Certaines grosses applications (comme des bases de données) requièrent des journées à installer, voire même le déplacement d'experts. Pour les applications plus petites mais déployées à grande échelle, les constats sont les mêmes : on peut citer le cas d'Adobe Reader 9 qui pèse 33 Mo mais qui nécessite le téléchargement et l'installation de Reader Download Manager qui occupe 210 Mo sur le disque dur et qui souvent reste stocké inutilement sur le disque dur après installation !

Dans cette évaluation, le mode de mise à jour est à prendre en

[56] http://download.intel.com/pressroom/pdf/cdsvsdownloadsrelease.pdf

compte. En effet, de plus en plus de logiciels vérifient automatiquement les dernières versions sur internet. Ceci crée une charge sur le réseau et surtout ralentit le démarrage des PC (car les check sont réalisés souvent au démarrage du PC). La maîtrise de ces mises à jour permettrait d'éviter une cause de l'obésiciel. Le mode de vérification des mises à jour doit donc être soigneusement choisi voir non mis par défaut. Cet impact est encore plus grand pour les appareils mobiles tels que les smartphones où les vérifications continuelles de mise à jour sont faites par les applications.

Diminution des éléments annexes et de la documentation

Impact des éléments annexes

L'étude Intel / Lawrence Berkeley cité dans le chapitre précédent montre qu'un logiciel peut impacter l'environnement à cause de son packaging. Cet impact n'est pas négligeable car cela nécessite des processus assez coûteux. Il est donc nécessaire d'étudier le cycle de vie de ces éléments et d'optimiser les impacts environnementaux. Dans ces éléments pouvant être associé à un logiciel, on peut lister :

- ✓ Boitier plastique
- ✓ Boite carton
- ✓ Blister plastique
- ✓ Documentations papier
- ✓ Publicités
- ✓ CD-ROM / DVD-ROM

Marketing et packaging durables

Ces éléments, si non nécessaires à l'utilisateur, sont à optimiser ou à supprimer. Ils mettent en effet en jeu des matériaux polluants (plastiques, papiers glacés…). Nous avons vu qu'une diffusion numérique maîtrisée était préférable. Il est cependant nécessaire dans certains cas de mettre en place un support physique de déploiement. La conception de ce packaging doit alors suivre les règles de l'éco-conception. Le marketing entre dans ce périmètre. L'aspect du boitier et la publicité doit donc être conçu en respectant les critères du développement durable :

- ✓ Le grammage et la taille des boites pourront par exemple être diminués,
- ✓ Des plastiques différents ne devront pas être mélangés pour permettre leur recyclage.

Pour aller plus loin

Guide du Marketing durable - Karine Viel, responsable du programme RSE du comité 21

Documentation

La documentation, après le packaging, est aussi non négligeable quand on analyse les impacts. Dans le cas d'une production papier de la documentation on doit être attentif au grammage et au type de papier pour limiter l'impact environnemental (respect des labels, taille des polices, grammage papier, …). Mais au même titre que le support du logiciel, une diffusion numérique de la documentation est préférable. Il est cependant important de choisir le bon format numérique. Le format html sera par exemple préférable à un support PDF (Compte tenu de l'impact plus faible). L'utilisateur

devra de plus être sensibilisé sur l'impression du support numérique afin de limiter l'impact d'une impression sur imprimante personnelle qui pourrait être plus polluante. Cette sensibilisation pourra se faire soit via des messages soit par des fonctionnalités d'impression optimisées. Une documentation en ligne est à utiliser avec prudence. En effet, l'impact sur le serveur et sur le réseau est très compliqué à évaluer, la documentation locale sera préférable. Une étude australienne de l'université de Melbourne a effectué une analyse sur le choix d'application locale ou dans le cloud[57]. Ces conclusions peuvent être appliquées aussi à la documentation. Les chercheurs ont identifiés un impact non négligeable du transport réseau entre la source du cloud et le client final. La réduction de l'énergie dépend donc du cas d'utilisation. Dans le cas d'une utilisation faible et non fréquente et de services conventionnels, le cloud peut être bénéfique. Cependant, dans le cas d'une utilisation plus importante, le transport réseau a un impact non négligeable sur la consommation. Les utilisateurs non professionnels ont donc intérêts à réaliser des tâches de routine sur des PC de faible consommation (PC portable, notebook…) et de réaliser des tâches nécessitant de la puissance de calcul et cela de façon non fréquente via le cloud. L'utilisation de PC consommant trop annule donc cet effet bénéfique. Le choix entre une documentation en ligne et locale est donc complexe. Une étude des utilisateurs cibles et de leurs habitudes sera nécessaire pour bien choisir le support. Cette étude sera aussi nécessaire pour connaître la nécessité ou non d'une impression de la documentation.

Dans le cas d'une impression de la documentation, l'optimisation du contenu permettra également de diminuer l'impact (taille de police et des images, nombre de langues prise en compte…). Le mélange entre impression et support numérique sera un axe d'optimisation : les éléments de base

---

57 http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F5%2F4357935%2F05559320.pdf%3Farnumber%3D5559320&authDecision=-203

pourront être imprimés contrairement aux aides avancées (car moins utilisées). Comme pour les boitiers, l'éco-conception devra être appliquée aux manuels (papier recyclé par exemple).

Dans l'analyse des choix à réaliser pour adapter le bon support papier / numérique, une étude portant sur la consultation d'une facture numérique en ligne (étude ADEME) par rapport à l'envoi pour consultation de la même facture papier montre qu'u delà de 3 minutes de lecture, le support papier est moins impactant.

MAINTENANCE

La maintenance logicielle et le développement durable

Introduction

La maintenance du logiciel est la phase qui suit la phase de développement et qui a deux objectifs : corriger les erreurs qui seraient détectées dans la phase de vie de logiciel et faire évoluer le logiciel pour qu'il soit toujours opérationnel :

- ✓ Maintenance évolutive technologique : s'assurer que le logiciel est toujours compatible avec l'évolution du matériel et des logiciels des couches basses

- ✓ Maintenance évolutive fonctionnelle : s'assurer que le logiciel est compatible avec les nouvelles contraintes réglementaires, évolutions du marché, attentes fortes des utilisateurs

Du point de vue développement durable, ces deux objectifs ont deux bénéfices principaux :

- ✓ Augmentation directe de la durée de vie du logiciel lui-même

- ✓ Augmentation indirecte de la durée de vie des systèmes auxquels le logiciel est connecté.

Pourquoi augmenter la durée de vie du logiciel ?

Selon Sun[58],

- ✓ 80% du coût du logiciel est dû à la maintenance

[58] http://www.oracle.com/technetwork/java/codeconv-138413.html

✓ Aucun logiciel n’est maintenu tout au long de sa vie par le même développeur

Ces contraintes mènent bien trop souvent à une maintenance insuffisante pour maintenir le logiciel et pour le conduire finalement à sa fin de vie. Au sens développement durable, ceci ne va dans le sens d’une prolongation de la durée de vie. Et alors me direz-vous ? Et bien plusieurs effets négatifs en ressortent :

✓ Le logiciel a tendance à devenir jetable

✓ Le redéveloppement du logiciel est souvent choisi et n’est pas en accord avec une idée de durabilité

✓ L’absence de durabilité et les contraintes économiques ne permettent pas une amélioration des aspects environnementaux.

✓ L’utilisateur n’a pas la possibilité de garder le logiciel sur une longue durée.

Augmenter la durée du logiciel même si cela semble contraignant offre donc pour l’utilisateur une possibilité de prolonger la durée de vie du logiciel et pour le logiciel la capacité d’intégrer des avancées en termes de diminution de l’impact environnemental.

Pourquoi augmenter la durée de vie des éléments auxquels est connecté le logiciel ?

Sans cette maintenance le logiciel serait laissé tel quel et ne prendrait pas en compte les évolutions de l’environnement extérieur. Le logiciel n’est pas un élément orphelin (une sorte d’ile) mais plutôt un élément connecté à un environnement hétéroclite : système d’exploitation, autres logiciels, matériel, utilisateur. Il est donc nécessaire de s’adapter constamment à

cet environnement, puisque les autres éléments évoluent indépendamment. Cette notion d'interdépendance est importante pour un développement logiciel durable.

Si la prise en compte des évolutions des éléments extérieurs n'est pas faite, des régressions peuvent se produire. On peut citer par exemple :

- ✓ Perte de performance sur une nouvelle plate-forme matérielle
- ✓ Incompatibilité avec des mécanismes de gestion d'alimentation d'une nouvelle version d'un OS
- ✓ Non compatibilité du logiciel avec de nouveaux logiciels

Au final, ces problématiques peuvent donner à l'utilisateur un sentiment d'obsolescence. Ce sentiment, que nous appellerons obsolescence ressentie, donne à l'utilisateur l'impression que le PC et les logiciels installés ne sont plus au goût du jour. Fausse conclusion car le logiciel est tout de même opérationnel mais plusieurs messages font penser le contraire : interface plus à jour, perte de performance, incompatibilité avec un logiciel qui lui est plus récent…

Ce sentiment, même si bénéfique pour l'industrie logiciel, ne va pas dans le sens d'une utilisation durable des logiciels.

Bonnes pratiques de codage

Améliorer la maintenabilité du logiciel va permettre de prolonger directement sa durée de vie d'une part et indirectement celle du système sur lequel il sera installé d'autre part. Cette maintenabilité dépend de chaque langage et de l'état de l'art des bonnes pratiques. Les règles de codage sont des guides intégrant ces bonnes pratiques. On retrouve généralement pour chaque langage des règles de codage.

Ces règles n'incluent encore pas des pratiques spécifiques au développement durable. Cependant elles jouent directement un rôle important sur l'amélioration de la durabilité du logiciel.

Guideline

Voici une liste de guideline officiels ou non. Il existe aussi des guideline pour des frameworks ou des projets spécifiques. Ces derniers guidelines sont très important car il couvre en plus de l'amélioration de la qualité et de la maintenabilité les aspects de modularité. En effet, comme nous avons vu chapitre « Efficacité d'architecture » p. 163, dans le cas de module extérieur à des applications, il est important de respecter une bonne intégration. Ces guidelines permettent d'éviter que les modules « polluent » les applications hôtes (comme Mozilla, Chrome…)

Idées d'action Green Code Lab :

Identifier dans ces guidelines les règles spécifiques éco-conception et celle en désaccord

Java : Sun a rédigé un guide de codage pour Java «Code programming for Java programming Language »[59]. Ce document contient les recommandations que Sun suit et conseille. Ces conseils couvrent le nom des fichiers, l'organisation des fichiers, l'indentation, les commentaires, les déclarations, les espaces, les conventions de nommage ainsi les bonnes pratiques de codage.

C : L'écriture du langage C a été normalisé par l'ISO (ANSI/ISO 9899-1990).

C# : Guildeline Microsoft [60] et Philips Healtcare[61]

[59] http://www.oracle.com/technetwork/java/codeconv-138413.html
[60] http://blogs.msdn.com/b/brada/archive/2005/01/26/361363.aspx

C++ : Google Style Guide[62]

Perl : Style Guide[63]

Python : Guideline de Python Enhancement Proposal (PEP)[64]

PHP : Guideline du framework Pear[65] ou pour le projet DRUPAL[66]

GNU/Linux : Guideline de codage du noyau[67]

Mozilla : Guideline pour les modules Mozilla[68]

---

61 http://www.tiobe.com/content/paperinfo/gemrcsharpcs.pdf
62 http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml
63 http://perldoc.perl.org/perlstyle.html
64 http://www.python.org/dev/peps/pep-0008/
65 http://pear.php.net/manual/en/standards.php
66 http://drupal.org/coding-standards
67 http://lxr.linux.no/source/Documentation/CodingStyle
68 http://www.mozilla.org/hacking/mozilla-style-guide.html

FIN DE VIE

Valorisation des éléments gérés par le logiciel

La fin de vie du logiciel peut paraître sans intérêt dans le cas d'un logiciel. Cependant comme tout produit dans l'éco-conception, il faut valoriser la fin de vie. La valorisation va permettre de remettre dans le circuit les matériaux et les éléments qui compose le produit et qui ont été générés pendant la phase de vie du produit. Dans un produit classique, on va par exemple démanteler le produit et valoriser les différents composants : récupération, incinération, refonte… Dans le logiciel, la valorisation est plus simple : hormis le packaging et le support de diffusion du logiciel qui suit les règles de recyclage de tout produit physique packagé, elle se résume à la suppression de tous les éléments numériques associés au logiciel.

Tout d'abord le développeur doit réfléchir aux éléments qui constituent le code. On peut lister les items suivant :

- ✓ L'exécutable stocké sur le disque dur ou sur mémoire non volatile
- ✓ Les fichiers nécessaires à l'exécutable (base de données, drivers…)
- ✓ Les fichiers installés dans le système d'exploitation et dans le système de registre
- ✓ Les données mise en mémoire
- ✓ Les éléments générés : fichiers temporaires…

La persistance des logiciels et des données associées est une des causes de l'obésiciel. De plus la mauvaise gestion des fichiers générés par le logiciel mène à une inflation des besoins en espace mémoire. L'hébergement de millions de site internet qui ne sont plus gérés mais toujours actifs est un exemple de cette inflation essentiellement en stockage physique.

Pour ce qui est des éléments qui constituent le code, une procédure de désinstallation doit prévoir la suppression de tous les fichiers et informations : Fichiers sources, clés de registres, menu, programmes résidents, fichiers temporaires, raccourcis automatiques, …. Cette phase est simple à mettre en place pour le développeur car il maîtrise normalement les éléments qui composent son logiciel. Il faut donc mettre des mécanismes qui évitent toute perte de cette maîtrise (Comme par exemple l’enregistrement du changement d’emplacement des fichiers d’installation par l’utilisateur)

Par sécurité, les logiciels préservent des données spécifiques : Clés de registres partagée, dll partagées, fichiers de configuration utilisateur... Des mécanismes doivent être intégrées (dans le logiciel ou alors dans le système d'exploitation) pour tracer cela et permettre à l'utilisateur de supprimer ultérieurement toutes les données. Une solution est d'intégrer cette fonctionnalité dans le système d'exploitation plutôt que dans le logiciel lui-même.

Pour les données générées par le logiciel, il faut différencier les les données maîtrisées par l'utilisateur et celles gérées par le logiciel. Les premières. ne sont pas couvertes par la fin de vie directe car elles sortent du périmètre du logiciel (fichiers bureautiques par exemple). Les données non maîtrisées sont-elles à prendre en compte. On fera la distinction en considérant que les éléments gérés par l’utilisateur sont les éléments qui sont des fichiers de sortie attendus par l’utilisateur. Si le logiciel doit générer des éléments comme des fichiers temporaires par exemple, il doit intégrer une fonctionnalité pour les supprimer au plus tôt ou proposer la purge, la suppression à l’utilisateur.

Cette pratique est facilement réalisable dans un logiciel seul mais plus difficile dans le cas d'un logiciel réparti (Navigateur web par exemple). Par exemple : si un utilisateur visionne un document quelconque à partir d'une page internet, le navigateur va stocker le fichier dans un espace temporaire pour l'ouvrir avec l'application dédiée. Une fois que l'application aura visualisé le fichier, le fichier restera dans l'espace de stockage. Cette fonctionnalité de proposition de purge doit donc être intégrée dans le navigateur.

Pour les éléments générés en mémoire, la problématique est à gérer en améliorant l'efficacité des données (voir chapitre « Green Patterns » p.163)

Valorisation des éléments gérés par l'utilisateur

Fichiers de sortie du logiciel

Les éléments gérés par l'utilisateur ne peuvent pas être valorisés dans la phase de la fin de vie car par définition ils sont des éléments de sortie qui deviennent la propriété de l'utilisateur. On ne peut par exemple pas supprimer les fichiers générés par une suite bureautique. Cependant, dans certains cas, il est possible de mettre en place des mécanismes qui aide à limiter l'impact de ces éléments.

Un des premiers mécanismes est l'archivage des éléments. Si le logiciel a une maîtrise sur les données générées par l'utilisateur, il est possible d'utiliser ce mécanisme. Par exemple les logiciels de messagerie offre ces fonctionnalité en proposant un archivage des courriels. La valorisation sera la plus bénéfique si elle est associée à une compression pour réduire l'empreinte mémoire.

Le deuxième mécanisme est la mise en œuvre d'une date d'expiration dans les fichiers générés. A l'issue de cette date

d’expiration, plusieurs actions sont possibles : proposition à l’utilisateur de supprimer le fichier ou de l’archiver. Ce mécanisme est utilisé dans la gestion des machines virtuelles de VMWare. On appelle cela le décommissionement des machines virtuelles.

Globalement, tout mécanisme qui permet de valoriser au mieux les éléments produits par le logiciel est une bonne pratique.

Des logiciels externes permettent d’effectuer un nettoyage de ce type de fichier. On peut citer par exemple Ccleaner. L’utilisation de ce type de logiciel a cependant le désavantage de se faire à l’initiative de l’utilisateur, n’est pas accessible à tout public et arrive toujours un peu trop tard dans le cycle de vie des fichiers en mode correctif. L’intégration dans les logiciels est donc à privilégier.

La valorisation des fichiers de sortie ne se résume pas simplement à la suppression des fichiers mais aussi à l’optimisation. La production des éléments d'un logiciel est une chose très importante. Maîtriser les éléments en terme de taille mémoire va permettre de réduire l'impact d'un document qui sera déployé, stocké, partagé par plusieurs personnes (par exemple des documents word).

L'application de certaines caractéristiques aux utilisateurs indirects permet de couvrir des problématiques importantes. Par exemple la caractéristique d'interopérabilité appliquée aux utilisateurs indirects (et donc aux fichiers de sorties des logiciels) amène à réfléchir à la capacité de générer des fichiers qui seront utilisables entre différents logiciels. Ceci amène à réfléchir dans les choix de développements ou d’achats de logiciels à donner une préférence aux logiciels qui prennent en compte des formats standards de sortie ou le respect d’une norme d’échange dans un secteur donné ou par défaut qui s’assurent de la possibilité d’intégrer des fonctionnalités d’import / export des données vers les autres formats du marché. Ne pas imposer une impasse technologique aux

utilisateurs sur les formats générés par une application est un facteur important à prendre en compte dans fin de vie d'un logiciel par un développeur.

La problématique des fichiers propriétaires doit être traitée car elle amène de nombreux problèmes : durabilité des logiciels (je suis obligé de changer de version de logiciel pour lire des nouveaux fichiers), Obésité des logiciels pour prendre en compte tous les types de fichiers...

Désinstallation des modules non utilisés

Quatre DSI sur cinq estiment que le coût des logiciels non utilisés sur le PC est de plus de 100 $ selon le rapport Software Efficiency Report commandé par 1E. Les disques durs sont effectivement remplis de programmes qui ne sont pas utilisés. Les raisons sont nombreuses :

- ✓ Logiciel installé de base avec la machine et non nécessaire pour l'utilisateur
- ✓ Logiciel installé par les administrateurs et aussi non nécessaire
- ✓ Logiciel plus utile pour l'utilisateur (obsolète, trop lourd…)
- ✓ Logiciel installé pour évaluation

Des logiciels comme AppClarity existent et permettent de gérer et supprimer les logiciels inutilisés. Comme pour les logiciels permettant de nettoyer les fichiers, ce type d'applications est une solution qui ajoute une surcouche. Traiter le problème dans le logiciel ou dans le système accueillant le logiciel est préférable. En effet, c'est le logiciel lui-même qui sait au mieux s'il est utilisé ou s'il est rarement utilisé. A la manière du profiling permettant de savoir quelles tâches du logiciel sont appliquées, il est possible au logiciel de connaître son utilisation. L'information ensuite disponible sur l'utilisation de

chaque logiciel peut être fournit au système d'exploitation ou à un autre système de gestion.

Ce principe est facilement applicable dans les systèmes comme les navigateurs internet (Chrome ou Firefox) qui intègrent potentiellement une multitude de module extérieur. Proposer la désinstallation à l'utilisateur permettrait de réduire l'obésiciel de telles applications. Aller plus loin en désinstallant le module sans demande serait possible. Dans ce cas, la réinstallation devra être presque automatique pour l'utilisateur. Ce fonctionnement pourra être presque transparent pour lui (installation via un programme stocké sur le disque ou par une connexion internet). On rejoint alors la bonne pratique de l'installation modulaire vu dans le chapitre « Diffusion » p.226

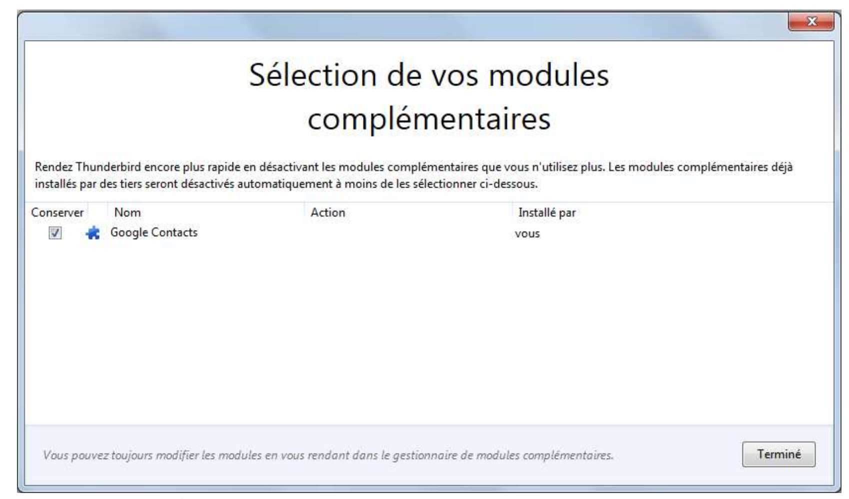


Figure 39 - Détection de non utilisation de module dans Thunderbird



6 Applications pratiques

Les parties précédentes sont un traité général de l'éco-conception des logiciels. Cette partie a pour but d'approfondir ces concepts en fonction des spécificités : domaines, métiers, technologies, langages…

APPLICATIONS WEB

Introduction

Nous entendons par applications Web toutes les applications qui utilisent l'infrastructure Internet, c'est à dire qui font intervenir un client sur un appareil indépendant (Mobile, PC, tablette...) et un ou plusieurs serveurs sur le Web. Puisque ces applications font intervenir plusieurs technologies et langages nous avons choisi d'en faire une partie spécifique.



Cette partie est en attente d'un spécialiste ou de personnes ayant quelque chose à dire ! C'est peut-être vous ? Contactez-nous sur info@greencodelab.fr si vous êtes intéressés.

Outils et logiciels

Mesure client

Green Fox

Module pour Firefox permettant de mesurer la consommation d'une page ou d'une application web. Ce module a été développé pour le challenge USI.

Firebug

L'outil Firebug sous Firefox permet une mesure du temps d'affichage des pages sur le poste client en décomposant l'ensemble des requêtes sur une page. Ceci est une aide très intéressante pour connaître les points de consommation important et les travailler. Les pages les plus souvent accédés comme les pages d'accueil, pages d'inscription, pages d'actualités peuvent être ainsi.

YSLOW

L'outil YSlow en ligne permet également une mesure de performance d'affichage d'une page et donne une notation en fonction des patterns décelées sur la page générée.

Recommandations pour un site Web

Ces recommandations sont principalement issues des travaux du groupe Green Software engineering. Elles ont été publiées aux éditions INSTICC[69].

Support du cache côté serveur

Le cache HTTP permet de réduire la quantité de requêtes HTTP et le volume de données transférées de manière significative. Les contenus qui sont générés dynamiquement par des scripts ou des programmes sur les serveurs ne sont pas habituellement conçus pour être mis en cache et donc pas marqués par des métadonnées de mise en cache par défaut.

Cependant, il peut y avoir des cas où le contenu rarement modifié change rarement. Par conséquent la mise en cache peut être possible sans risque de présenter des données obsolètes aux visiteurs.

Les systèmes de gestion de contenu (CMS) bien conçus peuvent être configurés pour utiliser des stratégies de cache des entêtes Expires et Last-Modified. Par conséquent, nous nous concentrons sur des sites Web ou des applications web

[69] Dick, Markus; Naumann, Stefan; Held, Alexandra: Green Web Engineering. A Set of Principles to Support the Development and Operation of "Green" Websites and their Utilization during a Website's Life Cycle. Filipe, Joaquim; Cordeiro, José (eds.): WEBIST 2010 : Proceedings of the 6th International Conference on Web Information Systems and Technologies, April 7 - 10, 2010, Valencia, Spain, Volume 1. Setúbal: INSTICC Press, 2010, pp. 48 - 55

qui ne sont pas exécutés via un CMS, mais sont mis en œuvre directement dans les scripts et programmes. Par exemple, pour une application Web qui enregistre les données des relevés de compteur et affiche des graphiques sur un site Web, la date Last-Modified peut être réglée à la date et l'heure où le dernier relevé a été inséré dans la base de données.

Nous recommandons que les développeurs Web implémentent le support des entêtes Expires et Last-Modified pour diminuer le volume de données transférées. Le code suivant montre comment cela peut être réalisé en script PHP. Un exemple plus sophistiqué est donné par Nottingham (2009).[70]

```
header ("Expires:". toGMT ($ expire));
header ("Last-Modified:". toGMT (lastmod $));
if (! isModifiedSince (lastmod $)) {
header ("HTTP/1.1 304 Not Modified ');
exit ();
}
echo 'CONTENU';
```

Dans un premier temps, l’entête Expires est fixé à une date qui est acceptable pour assurer la fraîcheur de la donnée. L'entête Last-Modified est définie avec une valeur qui indique la dernière modification. Après avoir réglé les en-têtes, l’entête Last-Modified-Since de la requête HTTP est comparé à la date de dernière modification du contenu. Si le contenu de la mémoire cache du navigateur est toujours valide, le statut HTTP non-modification est envoyé. Sinon, le contenu est envoyé avec un statut HTTP OK.

[70] Nottingham, M., 2009. Caching Tutorial. [Online] Available at: http://www.mnot.net/cache_docs/ [Accessed 10 Oct. 2009].

Minimiser JavaScript

JavaScript est largement utilisé sur les sites Web afin d'améliorer l'interactivité. Des fragments de code JavaScript sont souvent intégrés directement dans le HTML ou dans des fichiers JavaScript spécifiques. Les premiers augmentent la taille de la réponse, alors que les seconds augmentent la quantité de requêtes HTTP en particulier lorsque plus d'un fichier JavaScript est utilisé dans le site Web. Afin de minimiser la taille du contenu HTML, nous suggérons d'externalisation des fragments de code JavaScript embarqué dans HTML dans des fichiers JavaScript spécifiques. D'une part, cela se traduit par une demande supplémentaire HTTP, mais d'autre part, il permet:

- ✓ aux navigateurs de mettre en cache ces fichiers et de les utiliser dans les demandes ultérieures sans appeler le serveur d'origine
- ✓ de réduire la taille de téléchargement HTML surtout lorsque le contenu HTML ne peut pas être mis en cache parce qu'il doit être recalculé à chaque demande

Si le code JavaScript réside dans des fichiers dédiés, nous vous recommandons de définir des règles distinctes pour les entête Expire dans des configurations de serveur web, de sorte que ces fichiers expirent à l'avenir (en supposant que les changements de contenu HTML plus fréquents que framework technique).

Si plus d'un fichier JavaScript est utilisé dans un site Web, le nombre de requêtes HTTP peut être diminué si ces fichiers sont fusionnés en un seul fichier. Malgré l'augmentation du poids de fichier JavaScript, le nombre d'octets transférés sera un peu plus faible, car une seule requête HTTP doit être envoyée, et probablement revalidée lorsque les utilisateurs rechargent la page dans leur navigateur Web.

La taille du fichier JavaScript peut être considérablement minimisée en supprimant tous les caractères inutiles espaces ou des commentaires. Il peut encore être réduit en supprimant toutes les fonctions, méthodes ou des blocs de code qui ne sont pas nécessaires dans certain site Web. Cela est vrai pour les sites qui utilisent seulement quelques fonctions ou les méthodes de grosses bibliothèques JavaScript.

On pourra également s'assurer que les différentes bibliothèques JavaScript n'intègrent pas les mêmes fonctions ou méthodes qui alourdissent inutilement les pages ou que ces mêmes bibliothèques ne sont pas chargées plusieurs fois dans la même page.

Une autre technique est l'obfuscation. La plupart des outils disponibles appliquent d'abord les techniques d'optimisation et ensuite réduisent le code en substituant des variables, des méthodes ou des identificateurs de fonction et aussi des paramètres de méthode et fonction avec des noms plus courts. Les API de fonctions définies automatiquement et référencées dans le code HTML doivent bien sûr rester intactes.

prototype-1.6.0.3.js	Non compressé (KB)	Compression GZIP (KB)
Taille totale	126.70	28.49
Minimized size JSMin	93.09	22.71
Obfuscated size Dojo ShrinkSafe	87.66	24.94

Figure 40 - Minimisation, obfuscation et compression

Le tableau compare l'efficacité de l'outil de minimisation

JSMin[71] avec celle de l'outil d'offuscation ShrinkSafe (du Toolkit dojo http://www.dojotoolkit.org/) lors de l'optimisation de la bibliothèque JavaScript Prototype (http://www.prototypejs.org/). Comme on peut le remarquer, l'obfuscation est plus efficace que la minimisation dans le cas d'étude. Ceci résulte du fait que la minimisation ne remplace pas les identifiants de fonction avec des noms plus courts alors que la configuration par défaut de l'offuscation le fait. En contraste à la compression Gzip l'économie globale de la minimisation ou l'obfuscation sont moins efficaces (et dans ce cas particulier l'obfuscation est même contre-productif, car il y a moins de répétitions qui reste dans le code JavaScript et cela conduit à une meilleure compression). D'autres comparaisons avec les bibliothèques JavaScript conduisent à des résultats similaires (Dojo Toolkit, script.aculo.us).

Dans ce contexte, nous recommandons de minimiser JavaScript avec un outil comme JSMin et de compresser la réponse HTTP avec GZIP pour obtenir des transferts de données plus faibles. Nous vous recommandons l'obfuscation dans les scénarios où la compression ne peut pas être utilisée (par exemple si les clients HTTP ne sont pas entièrement compatibles avec les algorithmes de compression).

En outre, nous recommandons de minimiser ou d'obfusquer les fichiers avant qu'ils ne soient déployés sur le serveur au lieu de minimiser ou de les obfusquer dynamiquement à chaque requête HTTP pour gagner ainsi du temps processeur et de la consommation d'énergie.

Réduire et optimiser le CSS

Minimiser et optimiser les fichiers CSS offre de nombreuses possibilités de réduire les requêtes HTTP. Des économies de 40% à 60% sont possibles, si les opérations HTML ou JavaScript sont mises en œuvre avec du CSS. Grâce à une

[71] http://javascript.crockford.com/

mise en page conçu avec du CSS (King, 2008, p. 177)[72], il est possible d'économiser 25% à 50% de la taille des fichiers HTML (King, 2008, p. 180).

Le tableau suivant montre la taille du fichier d'une feuille de style CSS après optimisation et minimisation différentes.

Sans minimisation	2,812 Bytes
Avec minimisation standard	2,181 Bytes
Avec une minimisation élevée	2,040 Bytes
Avec une minimisation la plus haute	2,021 Bytes

Figure 41 - Minimisation

Ces résultats ont été obtenus avec CSSTidy (http://csstidy.sourceforge.net/). Cet outil utilise certaines techniques pour optimiser le code CSS. La minimisation standard a comme objectif de compression d'atteindre un équilibre entre la taille et la lisibilité. Avec cette réduction, le fichier est 631 octets plus petit et le taux de compression est de 22,4%. Une technique utilisée par l'outil est l'abréviation, par exemple avec la définition des couleurs. La définition de long {color:#ffcc00;} peut être écrit comme cela {color:#fc0;} .

Dans la technique de minimisation supérieure, la lisibilité doit être modérée. La taille du fichier a été réduite avec cette compression de 27,5%. En plus des autres techniques, les espaces inutiles, les caractères et les lignes vides sont supprimés. Cela peut diminuer la lisibilité, mais permet de

[72] King, A., 2008. Website Optimization. 1st ed. Sebastopol: O'Reilly Media.

réduire la taille du fichier.

La troisième optimisation est la minimisation la plus élevée. Son principal objectif est de parvenir à la plus petite taille du fichier. La lisibilité n'est pas considérée comme un critère. Le taux de compression est 28,1%. Dans ce cas, les wordwraps additionnel sont supprimés. L'ensemble du code est contenu dans une ligne.

Il y a d'autres techniques, qui permettent d'optimiser CSS :

- ✓ Un exemple est de remplacer les styles inlign avec les sélecteurs de type, de supprimer le code inutile ou d'utiliser des sélecteurs contextuels ou descendants au lieu de classes en ligne.

- ✓ Les déclarations d'éléments en double peuvent être évitées par l'utilisation de l'héritage.

- ✓ Certaines déclarations de propriété séparées peuvent être combinées dans une seule déclaration. Margin , border , font , background , list-style et les outline sont des raccourcis qui combinent des propriétés unitaires séparées.

- ✓ Les noms des classes ou des identifiants doivent être courts ou abrégés. Les noms longs sont uniquement importants pour les développeurs, mais pas pour les utilisateurs. D'autre part, les noms courts devraient être assez significatifs pour les développeurs.

Les déclarations CSS doivent être sauvegardées dans un fichier externe et ensuite incluses dans les fichiers du site. Ceci minimise la taille du fichier du document Web et les déclarations CSS seront chargées uniquement une seule fois et non à chaque requête du site.

Le CSS peut également être utilisé pour remplacer le code moins efficace. Un exemple est la définition de la table au

format HTML, qui peuvent être remplacés par du code CSS. Un autre point est l'utilisation embarqué JavaScript, comme 84,4% des pages web le font. Ces méthodes peuvent souvent être remplacées par du code CSS plus efficace. Les exemples sont les menus déroulants et l'effet de survol (King, 2008, p. 177).

Optimiser les éléments graphiques et des logos

54% du contenu d’un site classique est composé de graphiques (King, 2008, p. 157). Pour cette raison, l'optimisation des images est une voie importante pour minimiser les réponses HTTP.

Le logo, contrairement à une autre image, est normalement inclus plusieurs fois dans un site Web. Ainsi, l’optimisation du logo peut être plus efficace que l'optimisation d'une image, qui est principalement présenté une fois sur un site web.

Typiquement un logo n'est pas une photographie ou un tableau et en comparaison il est moins lourd. Il peut donc être sauvegardé comme un fichier GIF ou PNG. Les images PNG ont une meilleure compression. Pour des images en couleur, la taille est de 10% à 30% plus petite que pour les fichiers GIF (King, 2008, p. 169)

Il y a quelques possibilités pour optimiser un logo. Les plus efficaces sont :

- ✓ le changement du mode de couleur RVB vers une palette indexée
- ✓ la compression
- ✓ la sauvegarde avec des réglages d'économie spécifique pour internet
- ✓ le remplacement de texte graphique avec du texte CSS

formaté.

L'exemple suivant montre comment un logo peut être optimisé. Le logo se compose d'un objet graphique, un fond avec dégradé de couleur et de texte graphique.



Figure 42 - Le logo original avec des couleurs RVB

La première étape est de transformer les couleurs RGB en palette indexée. Dans ce cas particulier les couleurs peuvent être réduites à six. Le tableau suivant montre la taille du fichier après optimisation.

Logo original avec des couleurs RGB	16,959 bytes
Logo avec 16 couleurs	2,956 bytes
Logo avec 6 couleurs	1,984 bytes
Logo avec 4 couleurs sans texte	604 bytes

L'étape suivante est la transformation du texte graphique en texte CSS. Premièrement, le texte graphique doit être supprimé de l'image. Ensuite, le texte est inclus dans l'image avec du code CSS, qui définit une couche avec le logo comme image de fond. Les dimensions sont égales aux dimensions de l'image. Le code CSS est donc le suivant:

```
div.logo {
margin: 0;
largeur: 370px;
hauteur: 64px;
padding: 10px 70px;
police: 700 21px "Arial Narrow", "Arial", sans-serif;
color: # 000;
background-image: url ("logo.png");
background-repeat: no-repeat;
}
```

La taille du fichier peut ainsi être réduite de 16 959 octets à 604 octets (96% de réduction).

Des pixels uniques, des images couleurs, comme par exemple des images GIF, sont souvent utilisés pour créer des espaces de mise en pages. Ces images peuvent être remplacées par du code CSS, où l'espacement des cellules ont la même fonction que les images. Cela crée une mise en page similaire, mais à travers l'utilisation de CSS le téléchargement des images n'est plus nécessaire.

Les images qui sont relativement proches les unes aux autres, peuvent être combinées en une seule image. En outre, les liens vers chaque images simples peuvent être gardée avec une image mappée (King, 2008, p. 165). Une autre possibilité de combiner des images est d'utiliser des sprites CSS. Les sprites peuvent être très efficaces, si de nombreuses petites images comme des icônes sont sur un site web. La partie nécessaire de l’image sera affichée avec la règle background-position à la position désignée sur le site (Souders 2007).[73]

---

[73] Souders, S., 2007. High Performance Web Sites. Sebastopol: O’Reilly Media

Optimiser les photographies

La taille d’un fichier photographie est beaucoup plus grande que celle d'un logo. Par conséquent, il y a plus d'options pour réduire la taille du fichier. De façon générale, les images doivent être stockées avec des paramètres Web spécifique et les dimensions doivent être égales aux dimensions d’affichage souhaitées sur le site.

Une technique spécifique pour réduire la taille du fichier est de flouter l'arrière-plan. La technique de flou peut être utilisée pour les images, où certaines parties ne sont pas aussi importantes que d'autres (King 2008, p. 169).

La taille du fichier pour notre étude est de 126246 octets. Après le floutage de l'arrière-plan avec un niveau 8, la taille du fichier est seulement de 18 162 octets.

Optimisation des vidéos et des animations

Les fichiers multimédia comme les vidéos composent 98,6% des octets transférés sur Internet (King, 2008, p. 159). Ainsi, minimiser les fichiers multimédias est critique. Les techniques de base sont le montage avec les paramètres Web spécifiques, en utilisant les codecs spéciaux, la réduction du taux et l’utilisation d'équipement professionnel. D’autres fichiers multimédia comme les animations Flash peuvent également être réduits et optimisés. Dans ce cas, les images non optimisées et les trames d'animation trop nombreuses sont la cause de la taille trop importante. Les images doivent être optimisées avec des programmes de traitement d'image avant qu'ils ne soient importés dans Flash. Les animations doivent avoir une cadence réduite (King 2008, p. 170).

On peut également préférer quand cela est possible l’HTML5 aux animations flash très consommatrices sur le poste local notamment en CPU.

Configurer le support cache HTTP

La configuration du navigateur web de l'utilisateur ne peut pas être maîtrisée par les concepteurs Web ou des administrateurs, ils doivent se concentrer sur les configurations du cache côté serveur. Même si 80% des utilisateurs ont leurs navigateurs configurés pour la mise en cache, 20% ont toujours un cache vide (Theurer 2007)[74]. L'insertion de métadonnées de mise en cache par le serveur web va diminuer significativement la quantité de requêtes HTTP et les réponses HTTP. La mise en cache dans HTTP/1.1 est conçue pour réduire :

- ✓ la nécessité d'envoyer des requêtes aux serveurs (mécanisme d' « expiration »)
- ✓ la nécessité d'envoyer des réponses complètes aux utilisateurs (mécanisme de «validation»).

Le mécanisme de validation ne réduit pas la quantité de demandes HTTP, mais il réduit la charge utile des réponses HTTP qui sont envoyées vers le client et donc réduit la bande passante du réseau (Fielding et al. 1999, p. 74)[75].

Afin de faciliter le mécanisme d'expiration de serveurs HTTP, les administrateurs peuvent spécifier des entêtes Expires ou Cache-Control dans leur réponse. L'entête Expires comme décrit avec HTTP/1.0 (Berners-Lee, 1996, p. 41)[76] définit la

---

[74] Theurer, T., 2007. Performance Research, Part 2: Browser Cache Usage - Exposed! [Online] Available at: http://www.yuiblog.com/blog/2007/01/04/ performance-research-part-2/ [Accessed 10 Oct. 2009].

[75] Fielding, R.; Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T., 1999. Hypertext Transfer Protocol -- HTTP/1.1. Request for Comments 2616. [Online] The Internet Society. Available at: http://tools.ietf.org/html/ rfc2616 [Accessed 10 Oct. 2009].

[76] Berners-Lee, T., Fielding, R. & Frystyk, H., 1996. Hypertext Transfer Protocol -- HTTP/1.0. Request for Comments 1945. [Online] Network Working Group. Available at: http://tools.ietf.org/html/rfc1945 [Accessed 10 Oct. 2009].

date absolue après laquelle la réponse devrait expirée. Un problème mineur avec l'en-tête Expires est qu'il utilise une date explicite et string de temps et cela nécessite donc que les horloges du serveur et du client soientr synchronisées (Crocker, 1982[77], p. 26; Souders, 2007, p. 22[78]). Pour HTTP/1.1, cette limitation a été surmontée avec l'entête Cache-Control. Il utilise la directive max-age pour définir les secondes pendant lesquels la donnée demandée peut rester dans le cache. Pour rester compatible avec les clients HTTP plus ancien qui ne supportent pas le protocole HTTP/1.1, on peut définir l'en-tête Expires avec celui du Cache-Control. Dans ce cas, l'entête Cache-Control remplace l'entête Expires.

Le mécanisme de validation est utilisé par les clients HTTP qui ont une donnée entrante dans leur cache qui a déjà expiré. Dans ce cas, le client peut envoyer une requête conditionnelle HTTP au serveur. Cette demande conditionnelle HTTP ressemble exactement à une demande HTTP normale, mais en plus elle contient le «validateur» qui peut être utilisé par le serveur pour déterminer si la ressource demandée par le client est toujours à jour ou a besoin d'être rafraîchie. Dans le cas d'une actualisation nécessaire des données, une requête est envoyée au client, sinon le serveur répond avec le code de statut HTTP « 304 Not Modified ». Il y a deux validateurs qui peuvent être utilisés: la date Last-Modified ou les validateurs de cache Entité. Dans le cas de la date Last-Modified, une entrée de cache est considérée comme valable que si la ressource demandée n'a pas été modifié depuis la date Last-Modified. Une balise d'entité est un identificateur unique pour une version spécifique (Entity) d'une ressource (Fielding et al. 1999, p. 85). Le calcul des balises Entity dépend de la mise en œuvre du serveur web. Les spécifications HTTP/1.1 établissent que

[77] Crocker, David H., 1982. Standard for the Format of ARPA Internet Text Messages. Request for Comments 822. [Online] University of Delaware. Available at: http://tools.ietf.org/ html/rfc822 [Accessed 10 Oct. 2009].

[78] Souders, S., 2007. High Performance Web Sites. Sebastopol: O'Reilly Media.

les serveurs doivent envoyer les deux : les balises Entity et la date Last-Modified dans leurs réponses. Les clients HTTP/1.1 sont contraints par la spécification d'utiliser des balises Entity dans les demandes conditionnelles de cache s'ils sont fournis par le serveur. De plus les clients devraient appliquer la date Last-Modified si une avait été fixé (Fielding et al. 1999, p. 86).

Afin de réduire le montant total des requêtes HTTP ou la charge utile HTTP nous vous suggérons de configurer le support du cache client correctement. Cela signifie:

- ✓ fixer les dates d'expiration dans un avenir lointain et les entêtes CacheControl des ressources qui sont rarement changées

- ✓ les entêtes Last-Modified et des balises Entyty de toutes les ressources qui n'ont pas besoin de recalcul lors des demandes ultérieures (contenu principalement statique)

Un exemple simple de configuration du serveur web Apache peut ressembler à ceci :

```
Sur ExpiresActive
<FilesMatch "\.(html|jpg|png|js|css)$">
ExpiresDefault "Accès Plus 355 jours"
Taille MTime FileETag
</ FilesMatch>
```

La directive de configuration Apache ExpiresDefault gère à la fois, la génération d'une entête Expires et la génération d'un en-tête Cache-Control pour les types de ressource correspondant.

Utiliser la compression

Aujourd'hui, de nombreux navigateurs web modernes prennent en charge des compressions. La compression réduit la taille de la réponse et donc le temps de transfert, mais aussi la consommation d'énergie en raison de la petite taille des

données et des temps de transfert plus courts.

Les navigateurs Web supportent généralement le format de compression gzip ou DEFLATE. Les deux formats sont particulièrement cités dans la spécification HTTP/1.1. L'entête Accept-Encoding est utilisé par les navigateurs Web pour indiquer quels codages de contenu sont pris en compte. Un serveur Web peut compresser le contenu en utilisant l'une des méthodes de compression énumérés par le navigateur et doit aviser le navigateur dans l'entête de réponse Content-Encoding de la méthode de compression est utilisée (Fielding et al. 1999).

La compression n'est pas aussi simple qu'il y paraît, car il y a des versions de navigateurs plus anciens qui prétendent prendre en charge la compression, mais ne le font pas réellement, en raison d'incompatibilités ou de bogues. D'autre part, plus de 95% (ADTECH, 2008)[79] des navigateurs installés et utilisés en Europe sont connus pour prendre en charge la compression GZIP. Par conséquent, il est raisonnable d’activer la compression sur le côté serveur.

Cependant, tous les types de contenu ne sont pas adaptés à la compression, par exemple les formats d’images compressées, des fichiers de musique compressés ou les documents PDF. Compresser ces types de fichiers peut s’avérer même contre-productif. Ainsi, la compression doit être utilisée pour les fichiers qui sont bien compressibles comme des fichiers texte. Avec un site web simple exemple qui a été compréssé par la méthode GZIP avec un serveur web Apache, nous avons réalisé des économies comme le montre le tableau suivant. L'économie moyenne pour le site est d'environ 60% (en supposant que les images PNG ne sont pas compressées).

[79] ADTECH, 2008. Survey Unveils Extent of Internet Explorer Domination Across the European Browser Landscape. [Online] ADTECH: London. Available at: http://www.adtech.com/news/pr-08-07_en.htm [Accessed 10 Oct. 2009].

Exemple de contenu	Taille totale (KB)	Taille compressée (KB)	Economie
index.html	5,45	2,44	55,3%
style.css	2,73	0,68	75,1%
prototype.js	126,00	29,51	76,6%
ida-logo.png	24,80	24,86	-0,2%
UCB-logo.png	9,27	9,28	-0,1%

Figure 43 - Economies grâce à la compression

Green Challenge USI

Ecrit par Lionel Laské – C2S Groupe Bouygues

Le Green Challenge co-organisé par l'USI 2010 et par GreenIT.fr a réuni une quinzaine d'équipes d'avril à juin 2010. Son objectif: identifier des « greens patterns » de développement c'est-à-dire des bonnes pratiques logicielles pour réduire la consommation énergétique d'une application.

Rappel du périmètre du Challenge

Pour identifier les « greens patterns », le challenge proposait de faire baisser la consommation sur un exemple d'application: QRDecode. L'objectif de l'application QRDecode est de décoder des codes barre à deux dimensions (des « QR Codes »), d'afficher les coordonnées des contacts auxquels correspondent ces codes-barres et de positionner ces contacts sur une carte. Une implémentation de référence était proposée, en voici une capture d'écran.

Figure 44 - Affichage de l'application

Schématiquement, si l'on se limite aux flux, le rôle de cette application est de transformer plusieurs dizaines de QR Codes représentés par des fichiers .QRC transmis à l'application en:

✓ Des cartes de visites,

✓ Des images (la représentation visuelle du QR Code),

✓ Des coordonnées GPS,

✓ Une carte affichant ces coordonnées.

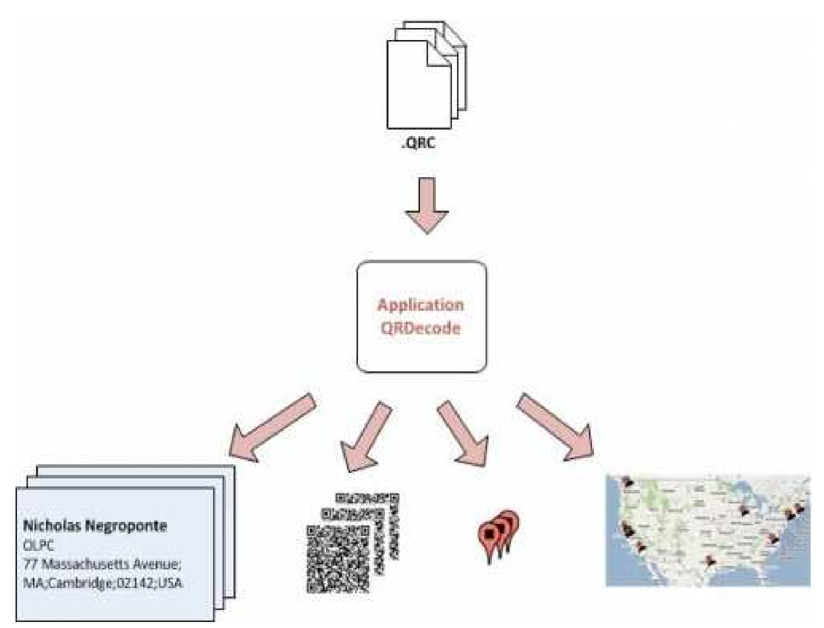


Figure 45 - Composition des QRcodes

Les différentes fonctionnalités de l'application QR Decode sont donc:

- ✓ Le décodage des fichiers QR Code en une représentation carte de visite (en l'occurrence VCard[80]),
- ✓ La transformation des QR Codes en images,
- ✓ La géolocalisation des adresses contenues dans les VCard pour obtenir des coordonnées GPS,
- ✓ Le positionnement des points GPS sur une carte graphique,
- ✓ L'affichage des cartes de visites et de la carte graphique.

L'implémentation de référence de l'application QR Decode a été réalisée en Java et se décompose en deux parties: une partie serveur qui s'exécute sur Google App Engine et une partie client qui s'exécute sur le navigateur. Les deux parties de l'application sont instrumentées pour récupérer le temps CPU consommé. Pour la partie serveur cela se réalise via des APIs spécifiques fournis avec le projet, pour la partie client cela se réalise par l'intermédiaire d'un plug-in FireFox développé pour l'occasion, GreenFox.

[80] http://fr.wikipedia.org/wiki/VCard

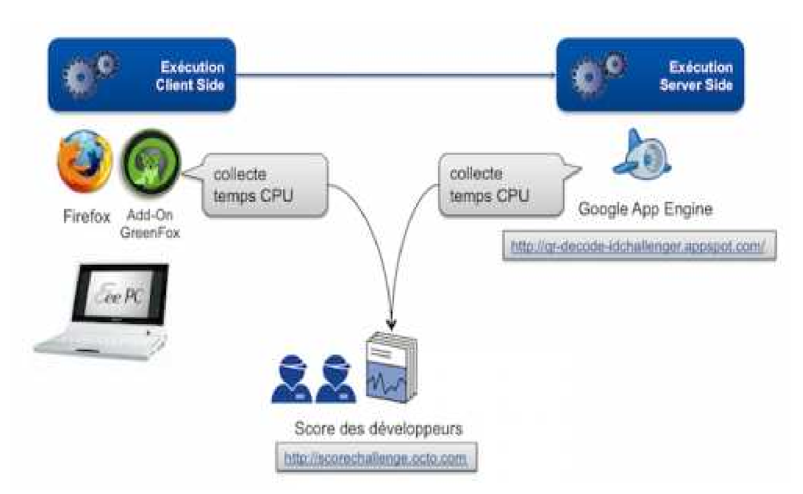


Figure 46 - Architecture de l'application

Les équipes étaient jugées en fonction de la réduction de la consommation qu'elles apportaient à l'application de référence. Les paragraphes suivant décrivent les méthodes mises en œuvre par les différents participants.

Répartition des traitements

La bonne répartition des traitements entre le serveur et le client est un des éléments majeurs d'optimisation des performances énergétiques de l'application. En effet, le code qui s'exécute côté serveur chez Google dispose de conditions énergétiques idéales (une plate-forme hautement mutualisée, un PUE optimisé) alors que le code qui s'exécute sur la machine cliente, même dans les conditions minimales que nous avions choisi (un netbook de type Asus EeePC) reste celui d'un PC individuel avec une importante déperdition d'énergie. Dans l'implémentation de référence, les traitements se répartissaient comme suit :

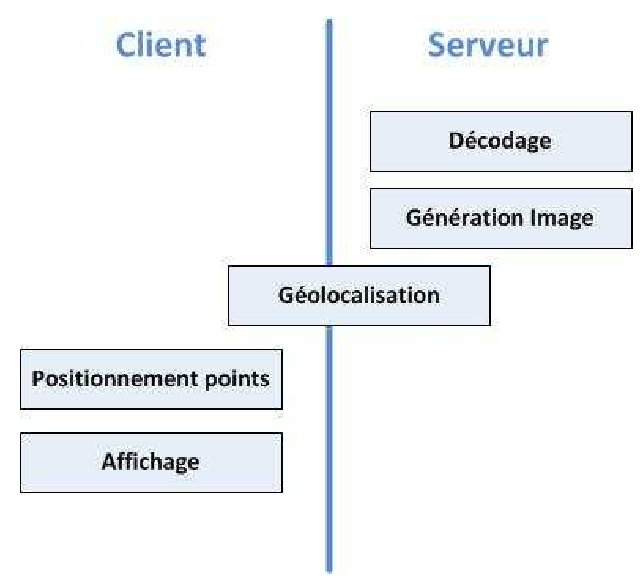


Figure 47 - Répartition des traitements initiale

La première optimisation consistait donc à décaler le plus possible de traitements côté serveur. Voici la répartition « idéale » proposée par les deux premières équipes sur le podium[81].

[81] https://sites.google.com/a/octo.com/green-challenge/resultats-du-challenge

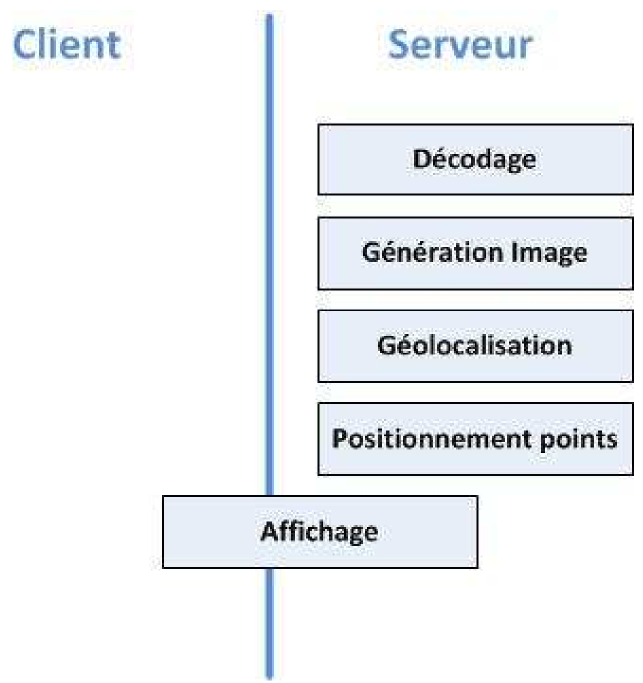


Figure 48 - Répartition modifiée des traitements

La **géolocalisation** est effectuée par appel d'un traitement Google. Le fait de le réaliser via un appel Javascript et nettement plus coûteux que lorsqu'il est intégré dans le code serveur. De plus, il est effectué pour chaque adresse ce qui est pénalisant. Une optimisation consiste à réaliser un appel batch sur le serveur pour réaliser la totalité du traitement en un seul appel. C'est possible via l'API Google mais cela impose des limites en nombre de requêtes lancées, un des gagnants a donc choisi d'utiliser MapQuest[82] à la place.

Le **positionnement** des points sur la carte est également réalisé dans l'application de référence par l'API Javascript de GoogleMap. C'est encore une fois très coûteux en CPU. Deux stratégies ont été utilisées pour éviter cet écueil. Dans les deux

[82] http://www.mapquest.com/openapi/

cas il s'agissait de supprimer les appels JavaScript. La première stratégie consiste à réaliser une génération complète sur le serveur, cela est possible en utilisant la version statique de Google Map qui génère une seule image intégrant la carte et tous les points.

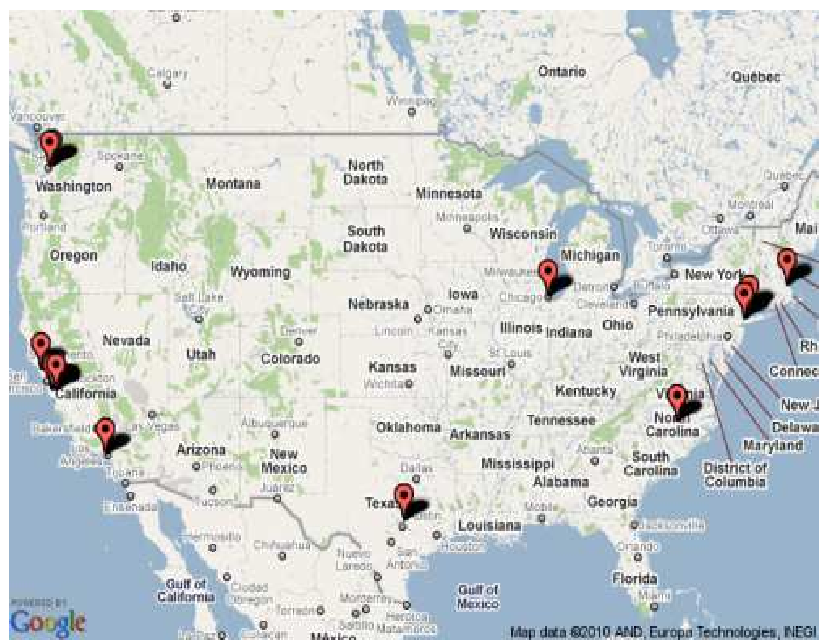


Figure 49 - Google Map statique

Une autre stratégie était l'utilisation d'un Canvas HTML 5 dans la page avec un positionnement des points en relatif sur un simple fond de carte.



Figure 50 - HTML5

Dans les deux cas cela limite les fonctionnalités de l'application ce qui était autorisé par le règlement.

L'**affichage** se fait dans le navigateur et est donc obligatoirement côté client. Néanmoins, la plupart des candidats ont choisi d'alléger les traitements d'affichage en préparant le travail côté serveur. Ainsi, deux des équipes gagnantes ont directement encodées les images dans la pageHTML (soit en base64 soit en passant par le data URI Scheme). Voir l'exemple ci-dessous.

<img height="50" width="50" src="data:image/png;
base64,iVBORw0KGgoAAAANSUhEUgAAADkAAAA5AQAAAA
CkY74o
AAACDElEQVR42gEBAv79AP///////+AAP///////+AAP/////
///+AAP//////+AAPAYd7eONAeAAPfbqk9pxfeAAPRQfik95Re
AAPRUm7PMLReAAPRXuYGD/ReAAPfVRdy43feAAPAVVVVV
VAeAAP
/0Zp05D+AAPQawgCpNB+AAPPkxdcGbK+AAP4JncYKlFeAAPN
m7P
7VX3eAAPTBYYKMZ9+AAPm+0JffuReAAPWb9RElkEeAAPl0j/6
V3
X+AAPpDjnDKRfeAAPP21YMOLO+AAP/ZrsBga3eAAPp1YvR7J
H+A
APedkyWKfbeAAPjzIuJHeQ+AAPIDg0C3cCeAAPRxGt1LRz+AAP
J
XDZTvJU+AAPJwKt0HZ3+AAPsE9cBTgDeAAPdmSEZypH+AAP
RNEJ
WpD5+AAPfwhKWPv/eAAPfFOPoUyOeAAP5v84t9lfeAAPSZk6K
AX
+AAP4j66af7u+AAPyOyI79UNeAAPy0CyhjnW+AAPfd4mu5JyeA
A
PonjJeeab+AAPuASpOR/aeAAPjynbVmw6eAAPHMC4C7IEeAAPz
p
NxW92+AAPAaVJQ/FXeAAPfS5pxXZ3eAAPRSTcGqIA+AAPRQ
HJKP
v6eAAPRV9gPpnN+AAPfY+dJQcPeAAPATFiaedyeAAP////////+
AAP////////+AAP//////+Ab4EaBY4WFZcAAAAASUVORK5CYII="
class="flashcode">

L’affichage HTML des cartes de visites a également été généré côté serveur ce qui permet d’éliminer le JavaScript qui génère le code HTML à partir de la représentation mémoire/JSON des données.

Optimisation des traitements

La plupart des équipes ont également travaillés sur l’optimisation des traitements côté serveur. Dans un premier

temps, l'idée était d'effectuer un profiling de l'application afin d'identifier les lignes de code sur lequel le processeur passe le plus de temps. Plusieurs outils Java ont été utilisés pour cela: Netbeans Profiler[83], Java VisualVM[84] ou JavaProfiler[85].



Figure 51 - Profiler

Voici les différentes optimisations réalisées :

Décodage

Le traitement de décodage des QR Code est clairement le traitement le plus coûteux en temps processeur sur la partie serveur.

Dans l'implémentation de référence, le traitement s'appuyait sur la librairie Open Source de Yusuke Yanbe[86]

[83] http://profiler.netbeans.org/
[84] http://java.sun.com/community/javavisualvm/
[85] http://code.google.com/p/java-profiler/
[86] http://qrcode.sourceforge.jp/



Figure 52 - Librairie Open Source

Deux des équipes sur le podium ont fait le choix de chercher et de benchmarker une autre librairie. Ils se sont donc appuyés sur la librairie Zebra Crossing[87] qui offre plus de fonctionnalités (support d'un plus grand nombre de type de code barre) et qui est surtout beaucoup plus performante que la librairie de départ.

Une autre stratégie, plus complexe, consistait à optimiser la librairie existante. Pour cela un profiling plus fin et des optimisations sur le code Java ont été mises en œuvres. En particulier:

[87] http://code.google.com/p/zxing/

- ✓ Eviter les copies de blocs mémoires (voir exemple de code ci-dessous),

- ✓ Limiter le nombre d'objets à instancier et favoriser la réutilisation des instances,

- ✓ Passer en variable Static des tableaux de valeurs,

- ✓ Limiter les conversions de types,

- ✓ Limiter l'utilisation d'objet nécessitant de la synchronisation (ThreadSafe).

A noter que l'objectif de ces optimisations n'est pas de limiter l'usage mémoire (peu impactant énergétiquement) mais de limiter le nombre d'opérations pour alléger la CPU.

```
int[][] imageToIntArray(QRCodeImage image) {
int width = image.getWidth();
int height = image.getHeight();
int[][] intImage = new int[width][height];
for (int y = 0; y < height; y++) {
for (int x = 0; x < width; x++) {
intImage[x][y] = image.getPixel(x, y);
}
}
return intImage;
}
```

Génération image

Un autre traitement coûteux était la génération de l'image graphique du QR Code à partir de l'adresse. Une implémentation était fournie dans l'application de référence. Deux équipes ont fait le choix de la substituer par un appel à une API GoogleChart qui propose cette fonctionnalité. Cela permet en un simple appel HTTP de disposer de l'image.

```
<img height="50" width="50"
src="http://chart.apis.google.com/chart?chs=228x228
&cht=qr&choe=UTF-
8&chl=BEGIN%3AVCARD%0D%0AN%3AChambers%3BJohn+T.
%0D%0ATEL%3BCELL
%3A%28408%29+524-7209%0D%0AADR%3BWORK%3A%3B
%3B170+W.+Tasman+Dr.%3BCA%3BSan+Jose%3B95134%3BUS
A
%0D%0AORG%3ACisco+Systems%3B%0D%0AEND%3AVCARD
" class="logo flashcode">
```

Même si cette option génère une économie énergétique, le jury a pris la décision de pénaliser ces deux équipes car l'appel à ce traitement n'est pas visible à travers nos instruments de mesures et pénalisait donc les autres équipes.
La plupart des autres équipes ont réalisés des optimisations sur le code de génération des images. L'optimisation la plus simple était de "rogner » la taille pour éviter le traitement de pixels inutiles.

Différentes optimisations ont également étaient réalisées sur le traitement pour éviter de manipuler des pixels de couleurs alors que l'image du QR Code est nécessairement en noir et blanc. Enfin, l'encodage de l'image en bitmap a été privilégié en évitant de passer par les APIs AWT qui sont peu performantes.

Affichage

L'optimisation de l'affichage consistait d'abord à éviter l'utilisation du JSP qui provoque un overhead surtout au premier chargement. La plupart des équipes ont également pris la décision de construire le code HTML directement en utilisant des StringBuilder (voir exemple ci-dessous).

```
public String decode(StringBuilder sbCards, StringBuilder sbAlerts,
StringBuilder sbImages, File file, int id) throws Throwable {
...
// HTML of vcard
sbCards.append("<li class = \"vcard\">");
// preparing the HTML5 canvas
sbCards.append("<canvas id=\"canvas");
sbCards.append(id);
sbCards.append("\" width=\"");
sbCards.append(wOut);
sbCards.append("\" height=\"");
sbCards.append(hOut);
sbCards.append("\"></canvas>");
sbCards.append("<span class=\"name\">");
sbCards.append(sName);
sbCards.append("</span>");
sbCards.append("<span class=\"orga\">");
sbCards.append(sOrga);
sbCards.append("</span>");
sbCards.append("<span class=\"addr\">");
sbCards.append(sAddress);
sbCards.append("</span>");
sbCards.append("</li>'+\r\n'");
…
```

La plupart des équipes ont aussi fait en sorte d'éviter les aller/retours qui nécessitent des traitements de connexions côté client et côté serveur. Une des équipes a fusionné dans la pageHTML: le code HTML, la feuille CSS, les images et le Javascript pour limiter les échanges à un seul aller/retour. Le code HTML a été optimisé (suppression des espacements inutiles) par plusieurs équipes. Le code JavaScript a également était optimisé pour limiter le nombre d'appel AJAX qui impliquent des traitements (et donc de la CPU) pour réaliser les connexions. Le JavaScript a aussi été offusqué pour limiter le parsing. La meilleure stratégie était néanmoins d'éviter complètement le JavaScript !

Autres optimisations

D’autres optimisations ont été mise en œuvre. La principale concerne la gestion des traces et du code de débogage. L’idée étant de réaliser ces traitements de manière conditionnelle pour ne pas consommer du temps d’exécution inutile. Par exemple le code de trace:

```
canvas.println("Adjust AP(" + x + ","+ y+") to d("+dx+","+dy+")");
```

est remplacé par:

```
if (canvas.isPrintlnEnabled())
canvas.println("Adjust AP({},{}) to d({},{})", x, y, dx, dy);
```

Enfin, plusieurs équipes ont mis en œuvre les options de gestion du cache côté navigateur qui peuvent être intéressantes si l’application s’exécute plusieurs fois. Néanmoins le jury partait systématiquement d’un cache vide avant chaque mesure.

Quels enseignements ?

Le premier enseignement que l’on peut tirer de ce challenger est que l’optimisation énergétique d’une application est une réalité. Les gains obtenus entre l’application de référence et les équipes gagnantes vont de 20 % pour la partie serveur à un gain de plus de 600 % sur la partie cliente.

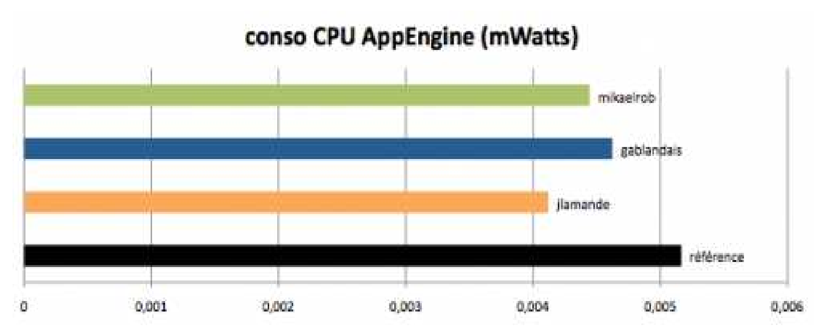


Figure 53 - Résultats côté serveur

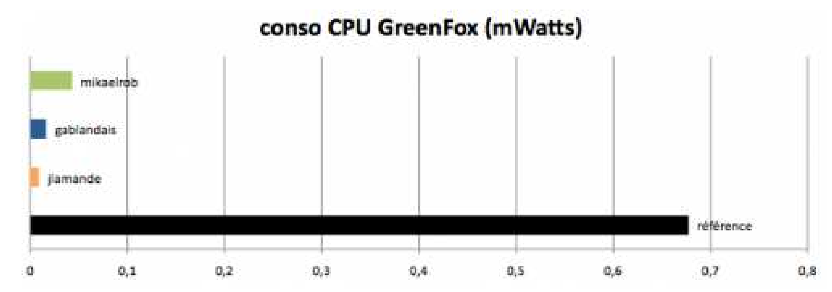


Figure 54 - Résultats côté client

On constate ensuite que les stratégiques gagnantes pour limiter la consommation sont:

- ✓ Réaliser le maximum de traitements côté serveur quitte à réduire l'ergonomie à l'essentiel côté client,
- ✓ Profiler l'application pour identifier les traitements fortement consommateur de CPU,
- ✓ Optimiser ces traitements pour limiter le nombre d'opérations réalisées ou remplacer ces traitements par des librairies plus efficaces,
- ✓ Pré-générer les affichages sur le serveur et éviter le code

interprété (JavaScript ou JSP).

C'est à ce prix que l'on aura des applications plus « green » mais aussi, globalement, de meilleure qualité.

APPLICATIONS MOBILES

Introduction

Les applications mobiles sont soumises à la contrainte de consommer le moins possible. En effet contrairement aux applications PC qui sont alimentées avec de l'énergie « illimitée », les appareils mobiles sont limités par l'autonomie de la batterie. Ce constat est d'autant plus important, que les appareils mobiles intègrent de plus en plus d'applications et de fonctionnalités.

Une étude IEEE[88] s'est penchée sur le design des applications mobiles et a montré une optimisation de l'efficience énergétique jusqu'à 45 % avec une augmentation de la performance de 30 %. Des principes intéressants sont à retenir :

- ✓ Réduire les interactions utilisateurs
- ✓ Éviter les vestiges des systèmes PC (scrollbars…)
- ✓ Utiliser la totalité de l'écran
- ✓ Cacher les longs et fréquentes actions utilisateur



Cette partie est en attente d'un spécialiste ou de personnes ayant quelque chose à dire. C'est peut être vous ? Contactez-nous sur info@greencodelab.fr si vous êtes intéressés.

[88] Keith S. Vallerio, Lin Zhong, and Niraj K. Jha. 2006. Energy-Efficient Graphical User Interface Design. IEEE Transactions on Mobile Computing 5, 7 (July 2006), 846-859. DOI=10.1109/TMC.2006.97 http://dx.doi.org/10.1109/TMC.2006.97

Outils et logiciels

De nombreux outils de mesure de la consommation existent. Ils sont cependant spécifiques à chaque technologies ou chaque mobile. Voici une liste de logiciels :

Android

Powertutor

C’est un logiciel qui permet de fournir la consommation des globale et des applications pour les téléphones Android. Des fichiers de log peuvent être enregistrés et rapatriés sur PC.

Nokia

Nokia Energy Profiler S60

Le Nokia Energy Profiler est une application qui permet aux développeurs et utilisateurs d'appareils d'examiner la consommation d'énergie et de plusieurs autres paramètres sur les appareils S60.

7 Annexes

GLOSSAIRE

Analyse du cycle de vie

Méthodologie d'évaluation d'un produit, d'un service ou d'une entreprise tout au long de son cycle de vie (« du berceau à la tombe »).

Bloatware

Tendance du logiciel à utiliser une quantité excessive de ressources ou de fonctionnalités.

Développement durable

(Source Wikipédia)

Nouvelle conception de l'intérêt public, appliquée à la croissance économique et reconsidérée à l'échelle mondiale afin de prendre en compte les aspects environnementaux et sociaux d'une planète globalisée.

Selon la définition proposée en 1987 par la Commission mondiale sur l'environnement et le développement dans le rapport Brundtland1, le développement durable est :

« Un développement qui répond aux besoins des générations du présent sans compromettre la capacité des générations futures à répondre aux leurs. Deux concepts sont inhérents à cette notion :

- ✓ le concept de « besoins », et plus particulièrement des besoins essentiels des plus démunis, à qui il convient d'accorder la plus grande priorité.
- ✓ l'idée des limitations que l'état de nos techniques et de notre organisation sociale impose sur la capacité de

l'environnement à répondre aux besoins actuels et à venir »

Eco-conception

Approche de conception des produits ou service prenant en compte les impacts environnementaux et les principes du développement durable.

Efficience

Qualité d'un rendement permettant de réaliser un objectif avec l'optimisation des moyens engagés. Autrement dit c'est l'efficacité de l'efficacité. (Source Wikipédia Efficience)

Efficacité

Capacité à réaliser une tâche.

Green Design Pattern

Motif de conception logiciel réutilisable qui réduit l'impact environnemental du système sur lequel il sera installé.

Obsolescence programmée

Réduction de la durée de vie du matériel pour des raisons techniques. Cette réduction peut être crée soit de manière volontaire lors de la conception du matériel ou soit par l'absence de possibilité de réparation des pannes.

Obsolescence ressentie

Réduction de la durée de vie du matériel pour des raisons subjectives. L'utilisateur a le sentiment que le produit est obsolète car il manque de performance (Bloatware) ou que la communication marketing communique sur des nouvelles fonctionnalités.

Obésiciel

Voir bloatware.

CREATIVE COMMONS LICENSE

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

a. "Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the

Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

b. "Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(g) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

c. "Distribute" means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

d. "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, Noncommercial, ShareAlike.

e. "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

f. "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the

case of broadcasts, the organization that transmits the broadcast.

g. "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

h. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

i. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the

communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

j. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights.

Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant.

Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

d. to Distribute and Publicly Perform Adaptations.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights described in Section 4(e).

4. Restrictions.

The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(d), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(d), as requested.

b. You may Distribute or Publicly Perform an Adaptation only under: (i) the terms of this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License

(e.g., Attribution-NonCommercial-ShareAlike 3.0 US) ("Applicable License"). You must include a copy of, or the URI, for Applicable License with every copy of each Adaptation You Distribute or Publicly Perform. You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License. You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.

c. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in con-nection with the exchange of copyrighted works.

d. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and, (iv) consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French

translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(d) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

e. For the avoidance of doubt:

    i. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

    ii. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(c) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,

    iii. Voluntary License Schemes. The Licensor reserves the right to collect royalties, whether

individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(c).

f. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING AND TO THE FULLEST EXTENT PERMITTED BY APPLICABLE LAW, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THIS EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN

NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further

action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.